1993

Comparative Study of Parallel Multipliers Based on Recoding Techniques

Vishal Abrol
University of Rhode Island

COMPARATIVE STUDY OF PARALLEL MULTIPLIERS BASED ON RECODING TECHNIQUES

BY

VISHAL ABROL

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL ENGINEERING

UNIVERSITY OF RHODE ISLAND

1993

MASTER OF SCIENCE THESIS

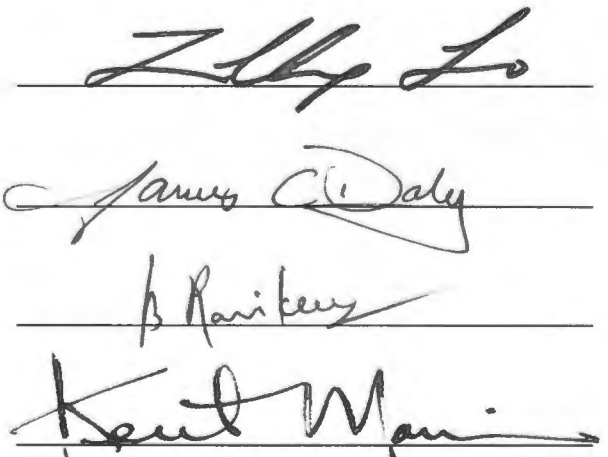
OF

VISHAL ABROL

APPROVED:

Thesis Committee

Major Professor



DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND

1993

Abstract

A 5-bit recoding scheme reduces the number of partial products by a factor of four in an array multiplier, but at the same time increases the complexity of recoding and partial products generation process. There is no concrete evidence yet, about efficiency or deficiency for applying 5-bit recoding. In this study, we compare the area-time performance of the VLSI implementation of 5-bit reoding array multipliers to its 3-bit recoding counterparts. Conditions in which a 5-bit multipliers is more efficient in terms of area and propagation time than that of a 3-bit recoding one are derived for given word lengths. We then present an example 5-bit recoding circuit design and its VLSI layout that yields the area-time efficiency.

Acknowledgments

First of all, I would like to thank my major Professor Dr. J. C. Lo for his guidance and cooperation in making this study a success and also recommending me for the financial support from the department. Thanks also goes to Dr. J. Daly and Dr. B. Ravikumar for agreeing to serve on my thesis committee.

Special thanks to my friends for helping me in using some of the software packages. Last, but not the least, I would thank all the instructors and the staff in the Department of Electrical Engineering at University of Rhode Island for making my stay at URI a memorable one.

Preface

This Thesis is prepared in accordance with the STANDARD PLAN outlined in the Guidelines for Thesis Preparation, Graduate School, University of Rhode Island, 1988.

Its main body contains an introduction and five chapters. Also included are, a Bibliography and List of References.

The Bibliography at the end of the thesis lists all possible sources used in the writing of this thesis report.

Contents

List of Tables

List of Figures

the main memory, specifies the operation to be performed by the arithmetic units and the address of the operands in the memory to be operated upon. After the numbers have been obtained from the memory, the arithmetic unit performs the operation specified by the instruction.

The extensive development of large scale digital computers in the past few years has naturally been accompanied by the corresponding development of the mathematical techniques required for the most efficient use of these tools of research. In the 70's and 80's, the multiplication in the low-end computer was usually performed on adder units. However, due to the advent of VLSI, the bulky multiplier can be realized on the processor chip. The multiplication speed is increased by this direct implementation. This also fits to the current trend of intensive scientific computations. For instance, important applications such as Digital Signal Processing, Image Processing, circuit simulation etc. depend on the speed of multiplication.

In many applications, the system performance depends upon the floating-point multiplication time. Speed of the integer multiplier has a direct effect on the speed of the floating-point multiplier, since the latter is built around the former. Designing a fast multiplier has been of great theoretical and practical interest to computer scientists and engineers. Various multiplication algorithms have been proposed and practically implemented.

The multiplication process can be divided into three different steps [2]:

- Adding each partial product to the accumulated sum.
- Shifting the multiplicand to the right.
- Adding the next partial product.

The above procedure repeats until all partial products have been added. Earlier, the number of ICs needed for multiplying two numbers varied from three to four. The 64×64 multiplier required four different ICs to implement the three steps of multiplication [3]. As the technology advanced, the implementation of a multiplier

was possible on a single IC.

Most of the early computers used fixed point arithmetic, but it presented difficulty in handling large scientific and engineering computations. Thus floating-point computation was proposed to overcome above difficulties, even though the operations which deal with data in scientific notations are more complex than fixed point or integer operations. Since, single precision multiplication is less accurate than the integer operation, therefore many floating-point operations are performed by double precision [4]. Most computers are equip with both fixed-point and floating-point arithmetic processor. Hence, there is a need for fixed-point and floating-point multipliers.

Conventionally, the Floating Point Unit (FPU) were external to the Central Processing Unit (CPU): a coprocessor [5]. The FPU was a coprocessor to the CPU. It included floating point multiplier, divider, ALU and register files. The coprocessor operates under integrated control with the CPU. The main coprocessor transmits the arithmetic operations sideways to be performed by the numeric processor. The results are then sent back when the processor needs them. Appearing in 1978, Intel's 8087 was the first coprocessor which performed the floating-point computation for both 8086 and 8088 [6]. Intel's 80287 which served as coprocessor for 80286 provided additional registers for the floating point computation. 80287/80387 are the coprocessor support for 80386.

The current trend is toward putting FPU on the processor chip. For example, Intel 80486 integrates the numeric coprocessor and CPU on a single chip. The advent of RISC processors also brought CPU core, cache memory and FPU on one chip. Since, the RISC concept greatly reduces the complexity of the processor, and at the same time, the advent of VLSI increases the number of devices per VLSI chip, the demand for high performance floating point coprocessors has created a need for high-speed small area-multipliers.

The multiplication of unsigned numbers using one's complement format and two's complement format are treated differently. Until 1951, the result of the multiplication of two numbers had to be shifted, if either or both of the operands were negative.

Booth [7] derived an algorithm which produced the result without any need for the correction. The use of two's complement numbers significantly simplified the addition process. Ghest [8] described a multiplier chip, which implements Booth's algorithm in a parallel mode. This algorithm produces a 2's-complement product when multiplier and multiplicand are represented in 2's-complement form. This was the first practical high speed multiplier.

The processing revolution in digital computers has been augmented by even more dramatic advances in microelectronic circuitry in silicon. One of the performance limitations of the multipliers has been governed by the VLSI technology. Bipolar technology has been promising high speed approach for decades. The advent of MOS technology promised higher levels of integration. The latest MOS circuits have the capability of high speed operation. The MOS field has three possible streams: nMOS, pMOS and CMOS devices. Each of these fields has a property which makes it attractive in one way or the other. The CMOS has its own share of limitations. There are some speed constraints in the CMOS technology. Also, Gallium Arsenide (GaAs) has recently been investigated for future high-speed technology needs. Howéver, compared to CMOS, GaAs and Bipolar have higher power dissipation and suffer from low yield problems.

It has been a challenging task to develop a multiplier, which can perform multiplication in time, proportional to the logarithm of the word length of the operands, and has a regular cellular array structure suitable for VLSI implementation. However, comparing the performance of different multiplier designs is extremely difficult. For instance, when a multiplier design claims a speed of 10ns, there is no indication that this is a better design than the one with a speed of 15ns. Of course, we must first make sure that they are both of the same word length. Still there are too many variables in the domain of multiplier design. In this example, the 10ns multiplier may be a GaAs array multiplier, while 15ns one may be a CMOS Booth multiplier. The new submicron technology helps to achieve few of the above requirements [9].

This work essentially points out the constraints in using different techniques for multiplication. The technology used here is not of great importance, since technology

alone cannot decide the best multiplier. The emphasis here is to compare the 3-bit and the 5-bit recoding schemes. There is no literature that justifies a better 5-bit recoding over a 3-bit recoding. This has necessitated to do the evaluation of different procedures in multiplication to support the argument. Through this comparative study, we shall establish means for unbiased performance comparison of multiplier design. The background description of the multipliers, and the steps needed to carry out the multiplication process are discussed in Chapter 2. The design of the recoder and the area of the multiplier is calculated, using different techniques, based on the trial layouts in Chapter 3. The speed evaluation which points out the area-time factor in any design is highlighted in Chapter 4.

This comparative study is qualitative because other approaches use different technologies, algorithms and possible margins (temperature range etc.). The study is done to refute that any particular combination for multiplication cannot be claimed as the best technique.

Chapter 2

Background Theory

Integer multiplication is inherently slower than integer addition or subtraction. However, by design technique, the multiplication in hardware implementation can be improved [10]. Multiplication is governed by two different processes. The crux of this multiplication process is the generation of the partial products and the summation of partial products [11]. The "add and shift" approach is the straightforward technique to perform a multiplication when only adding and shifting resources are available. For example, multiplying two unsigned binary numbers is performed as follows:

$$
\begin{array}{cccccc}
 & & & 1 & 0 & 1 \\
 & & & 1 & 1 & 0 \\
\hline
 & & & 0 & 0 & 0 \\
 & & 1 & 0 & 1 & \\
 & 1 & 0 & 1 & & \\
\hline
0 & 1 & 1 & 1 & 1 & 0
\end{array}
$$

When this operation is simulated in a software routine, each bit of the multiplier results in one add and one shift operation. Since most computers can add and shift in the same instruction cycle, at least n operations are required for n bits of multiplication. This sequential method is adequate for indirect implementation of multiplication. For high speed multiplication a combinatorial approach is needed.

In this approach partial products are formed simultaneously and then added concurrently. Each bit in the partial products is formed by ANDing the multiplicand bit with the multiplier bit [12]. While the partial product generation is fast the summation of partial products is slow.

There are many methods to speed up the multiplication process. At the first stage, the speed up can be accomplished by employing modified Booth recoding or k-bit recoding [11] techniques to reduce the number of partial products. This ultimately reduces the partial product summation time. The speed up for the summation of partial products may be achieved by using carry-save adder, Wallace tree [13], counters [14] and 4:2 compressor [15]. At the final stage of adding the final two partial product can be further reduced [3]. At this end conditional-sum, carry-select [16] and carry-lookahead adders are used to reduce the multiplication time. For a certain size of word length carry-lookahead has been proved to be the fastest adder [12]. Thus carry-lookahead is employed in the last stage for addition. Instead of waiting for the carry to ripple, carry is added at a later stage. For a $n \times n$ array multiplier, n partial products are generated. This means an array of $O(n^2)$ devices. Different circuits, structures and physical implementation of partial product reduction array portion of the multiplier were analyzed in [17].

2.1 Generation of Partial Product

In multiplication, if one addition is performed for each *one* in the multiplier, the average multiplication would require half as many additions as there are bits in the multiplier. This can be improved considerably by the use of both addition and subtraction of the multiplicand. The rules of determining when to add or subtract were developed, and the method of determining the number of operations to expect from the bit groupings was explained [11]. This resulted in a variable number of add cycles for fixed length multipliers.

Substituting shift cycles for add cycles when the multiplier bit is *zero* can reduce

the addition time. The time can be reduced further if it has the ability of shifting more than one position at a time when there are several *zeros* in a group. Assuming random distribution with equal numbers of *ones* and *zeros* in the multiplier, this should result in a 50% reduction in time. Skipping the 0's in a string is quite clear; skipping the 1's uses a special property of the string. A binary integer was represented in the following form:

$$X = A_n 2^n + A_{n-1} 2^{n-1} + \cdots + A_1 2^1 + A_0 2^0. \tag{2.1}$$

The actual number as written would consist of characteristics only and would be written as $A_n A_{n-1} \cdots A_1 A_0$. If such a number contained the coefficients ...01111111110.., this part of the number would have the value $2^{n-1} + 2^{n-2} + \cdots + 2^{n-x}$, where n is the position number of the highest order *one* in the group for which the lowest order position in the number is designated *zero*, and x is the number of successive *ones* in a group. The numerical value of this last expression may also be obtained from the expression $2^n - 2^{n-x}$. Thus for any string of *ones* in a multiplier the necessity of one addition for each bit can be replaced by one addition and one subtraction for each group. The only additional equipment that is required is a means of complementing the multiplicand to permit subtracting. A shift counter or some equivalent device was provided to keep track of the number of shifts and to recognize the completion of multiplication.

The above method of multiplication is dependent on the number of *ones* in the multiplier. Multiplication using uniform shifts is considered more practical. The number of shifts over here is dependent on the size of the multiplier rather than the number of *ones*. The number of cycles can be predicted using uniform shifts.

Rules were developed for handling two-bit and three-bit multiplier grouping. The discussion of multiplier is incomplete without mentioning the name of Booth. Booth [7] described a technique which uses recoding of binary numbers for multiplication. The purpose of a Booth's algorithm is to skip over a string of bits rather than to form a partial product for each bit. This process was independent of any foreknowledge of the signs of these numbers. Since the two's complement representation of numbers

is used almost universally, Booth's algorithm treated positive and negative numbers alike. Booth's algorithm scans two bits, at a time, of the multiplier number X and generate partial products in the form of -Y, 0, or +Y.

- If $x_i, x_{i-1} = 00$ or $x_i, x_{i-1} = 11$, the partial product is shifted one bit to the right.
- If $x_i, x_{i-1} = 01$, the multiplicand Y is added to the existing sum of partial product and result is shifted one bit to the right.
- If $x_i, x_{i-1} = 10$, the multiplicand Y is subtracted from the existing sum of partial product and result is shifted one bit to the right.

The procedure requires the attachment of the dummy digit(x_{-1}=0) to the right of L.S.B. and then evaluating the binary pair. In two-bit recoding each adjacent pair share one bit, so that in every iteration only one bit of the multiplier retires. In this way an n-bit multiplier generates n partial product.

Another speed up technique, a modification of Booth described by MacSorley [11] increases speed by reducing the number of partial products by a factor of two; this reduces the number of CSA stages and the gate count. The modified Booth's algorithm reduces the number of partial products by half, because the modified Booth scans three bits at a time. Each multiplier bit is divided into the substrings of 3 bits each, with adjacent bit sharing a common bit. In every iteration, two bits of the multiplier will retire. Apparently, the summation time for all partial products is significantly reduced. The array multiplier implementation has a complexity of $O(\frac{n^2}{2})$. A proof of Modified Booth algorithm was later given by Rubinfield [18]. However the partial product generation is not trivial. Table 2.1 shows the signed digit value corresponding to 3-bit recoding for all possible values in radix 4.

MacSorley had also mentioned 4-bit recoding scheme for multipliers. Here 4-bits are scanned at a time for generating the partial products. Table 2.2 shows the values corresponding to 4-bit recoding. This method is straightforward and easy to implement. Y and 2Y can be easily generated by shifting the string of Y bits

Table 2.1: Radix 4 representation

Triplet Value	Signed Digit Value
000	0
001	+1
010	+1
011	+2
100	-2
101	-1
110	-1
111	0

by 0 and 1 to the left. The problem is in generating the signed digit value 3Y. A carry-select adder was proposed for generating 3Y. This can be done by adding Y and 2Y. The idea of using 4-bit recoding was abandoned as the improvement in delay was insignificant compared to the additional area required. Moreover, 8, 12, 16 and 32 bits cannot be recoded uniformly without the addition of a bit to the left of the string. Nevertheless, it is a good choice for some moderate improvement in the speed of the multiplier.

Modified Booth recoding was later extended to recode 5-bits and higher [19]. Table 2.4 shows the signed digit value for the 5-bit recoding. The multiplier is recoded in 5-bit groups. This corresponds to radix-16 representation (k=4) for X using signed digits $0, \pm 1, \cdots, \pm 8$. Each recoder cell is larger than before but only n/4 of these are required compared to n/2 for modified Booth algorithm. The increase in the number of recoding bits will lead to significant reduction of the number of partial products, but, at the same time adds to the complexity of partial product generation. The generation of odd multiples of Y (3Y, 5Y, 7Y) is difficult in this form of recoding. All the other multiples (Y, 2Y, 4Y, 8Y) are generated by shifting Y (i.e. hardware displacement) to the left by 0, 1, 2, or 3 bits respectively. 6Y is obtained by displacing 3Y one position to the left. In [19] the handling of the odd multiples was done by using carry-select adders (i.e. power of two multiples to be added) to obtain the required multiples. There is a need of three such adders to obtain the odd

Table 2.2: Radix 8 representation

Quadruplet Value	Signed Digit Value
0000	0
0001	+1
0010	+1
0011	+2
0100	+2
0101	+3
0110	+3
0111	+4
1000	-4
1001	-3
1010	-3
1011	-2
1100	-2
1101	-1
1110	-1
1111	0

multiples of Y according to Table 2.3. Each row has $n + 3$ selectors selecting one of the nine multiples (0Y through 8Y) with the sign selecting addition or subtraction. The 5-bit recoding was claimed to be an attractive choice for large multiplier sizes (48×48), because of the high speed performance and at the same time good area cost [19]. However, the results shown in [19] may be misleading, since there is no actual comparative study that exists.

Typically for a k-bit recoding $\frac{n}{k-1}$ partial products are generated. However, these partial products are in the form of $-2^{k-2}Y, -(2^{k-2} - 1)Y, \cdots, 0, \cdots, +(2^{k-2} - 1)Y, +2^{k-2}Y$. The main advantage of going to a higher bit recoding lies in its reducing the CSA array delay. By doubling the size of k, the delay of CSA array is halved. The most formidable obstacle in going to a higher bit recoding is generation of odd multiples of the multiplicand e.g. 3Y, 5Y, 7Y, etc. Every time the recoding size is increased by one bit, the number of digits required to represent the binary number is doubled. Thus increase in one bit requires selectors, which should select from twice as many digits as before. Moreover, the number of odd multiples of the multiplicand is doubled requiring more carry-select adders to implement those multiples. There are some multiples of Y which can be obtained by addition of more than two power-of-2 multiples. The odd multiple 11Y is obtained by the addition of 8Y, 2Y and Y. Such multiples can be formed by using carry-save adders to reduce the summands to two numbers and then by a carry-select adders to obtain the required multiple. Thus there is a need of additional carry-save adder apart from the carry select adder for generating the odd multiples for a higher bit recoding. The hardware size increases exponentially as the value of k increases. The use of 9-bit recoding would offer only 25% improvement in the array delay over the 5-bit case, while requiring 1 of 128 selectors and the generation of 63 odd multiples of Y [20]. Then the complexity determined by the number of devices will no longer estimate the actual implementation complexity accurately. For VLSI implementation, the distribution of all the partial products formation to the partial product summation array will take a fairly large area proportional to k. Thus, the operations required, and the number, and the complexity of circuits needed, to implement them, grows very quickly, and soon

Table 2.3: Adders needed for the multiplier

Multiple	Operation	Adder Size
3Y	Y+2Y	m+1
5Y	Y+4Y	m+2
7Y	8Y-Y	m+3

Table 2.4: Radix 16 representation

Quintuplet Value	Signed Digit Value	Quintuplet Value	Signed Digit Value
00000	0	10000	-8
00001	+1	10001	-7
00010	+1	10010	-7
00011	+2	10011	-6
00100	+2	10100	-6
00101	+3	10101	-5
00110	+3	10110	-5
00111	+4	10111	-4
01000	+4	11000	-4
01001	+5	11001	-3
01010	+5	11010	-3
01011	+6	11011	-2
01100	+6	11100	-2
01101	+7	11101	-1
01110	+7	11110	-1
01111	+8	11111	0

prohibits a practical multiplier design using a higher bit recoding.

2.2 Recoding Algorithm

The mathematical representation of partial products has been derived to consummate the discussion of recoding. Consider the case of unsigned numbers; let X represent the multiplicand, and Y=$Y_{n-1}, Y_{n-2}, \cdots, Y_1, Y_0$ an integer multiplier - the binary point following Y_0. The lowest action is derived from the multiplier bits $Y_1, Y_0, 0$. The next higher action is found from the multiplier bits Y_3, Y_2, Y_1, but the resulting action is shifted by 2. For the highest order action with an unsigned multiplier the action must be derived with a leading or padded zero. For an odd number of multiplier the last action will be defined by $0, Y_{n-1}, Y_{n-2}$. Multipliers in the two's complement form may be used directly in the algorithm. In this case the highest order action is determined by $Y_{n-1}, Y_{n-2}, Y_{n-3}$ for an even number of multiplier bits and $Y_{n-1}, Y_{n-1}, Y_{n-2}$ a sign extended group for an odd sized multiplier.

Given an (n+1) digit binary vector B = $B_n, B_{n-1}, \cdots, B_1, B_0$, to obtain (n+1)-digit canonical **SD** vector D = $D_n, D_{n-1}, \cdots, D_1, D_0$ with $D_i = \{\bar{1}, 0, 1\}$. This process can be extended to generating two or even more signed digits at a time [12]. The radix-4 **SD** vector will have the following digit set.

$$\{\bar{2}, \bar{1}, 0, 1, 2\}$$

$$X = \sum_{i=0}^{n/2-1} (D_i.4^i) \tag{2.2}$$

An SD representation of X in radix 2^k will have n/k signed digits. The value of X can be written as

$$X = \sum_{i=0}^{n/k-1} 2^{ki} D_i, \tag{2.3}$$

where

$$D_i = x_{ki-1} + \sum_{j=1}^{k-1} (2^{j-1}.x_{ki+j-1}) - x_{k(i+1)-1} 2^{k-1}, \tag{2.4}$$

The proof of generalized multibit recoding was given by H. Sam et.al. [19]. After substituting the values of D_i in equation (2.2) and rearranging

$$X = \sum_{i=0}^{n/k-1} x_{ki-1}.2^{ki} - x_{k(i+1)-1}.2^{ki+k-1} + \sum_{i=0}^{n/k-1}\sum_{j=1}^{k-1}(2^{ki+j-1}.x_{ki+j-1}) \tag{2.5}$$

The above equation can be represented in n-bit two's complement binary integer format as

$$X = -x_{n-1}2^{n-1} + \sum_{i=0}^{n/k-1}\sum_{j=0}^{k-1} 2^{ki+j-1}.x_{ki+j-1} \tag{2.6}$$

2.3 Partial Product Reduction

The partial products that are generated using different schemes are to be added to form the required product of the two n bit numbers. Different hardware units can be employed to add these partial products. Each partial product forms a row in the multiplication array. Each bit in the row is represented by a weight. These weights have the values ranging from $2^0, 2^1, \cdots 2^n$, depending on the position of the bit. When a row has to be added to the other row, bits representing the same weight have to be added. The number of rows that can be added depends on the complexity of the adding unit. This adding unit is a compressor which receives inputs and generates outputs. The number of compressors required is a function of the number of columns and rows in an array. This reduction of partial products is done to accelerate the addition of summands. The techniques described for the reduction of partial products are discussed in the following subsections.

2.3.1 Array Multiplier

This is the most basic form of multiplier. The attractive feature of this multiplier is its regular array structure. Such arrays lead to a design, that is easy to lay out efficiently, and have a high throughput. Figure 2.1 shows an array multiplier of size $n \times n$ proposed by Braun [21]. It needs $n(n-1)$ full adders and n^2 AND gates to

implement it. Consider two unsigned binary integers $A = a_{m-1} \cdots a_1 a_0$ and B = $b_{n-1} \cdots, b_1 b_0$ with values A_v and B_v, respectively. In a binary multiplication, the (m+n)-bit product $P = P_{m+n-1} \cdots, P_1 P_0$ is formed by multiplying the multiplicand A by the multiplier B. The product P has a value

$$P_v = A_v B_v = (\sum_{i=0}^{m-1} a_i 2^i)(\sum_{i=0}^{n-1} b_j 2^j) \tag{2.7}$$

$$= \sum_{i=0}^{m-1} \sum_{i=0}^{n-1} a_i b_j 2^{i+j} \tag{2.8}$$

$$= \sum_{k=0}^{m+n-1} P_k 2^k \tag{2.9}$$

Each of the partial products $a_j b_j$ is called a summand. The arrangement of these summands form a matrix layout. For the operation involving other than unsigned multiplication, complementers have to be included. These operations are typically sign-magnitude, one's complement, and two's complement multiplication. The pre-complementer will convert the two operands to the positive integers before they can be multiplied by the core of unsigned multiplication array. The post complementer will convert the result back to the signed number representation if the two input operands do not agree in their signs.

This form of multiplier employs half adders for the top most row and full adders for the remaining part of the array to form the final product. Assuming that the numbers are in two's complement form, the time delay of this n-bit multiplier can be calculated as follows:

$$\Delta_{TC} = (2n - 2)\Delta_{FA} \tag{2.10}$$

Although it has a very regular structure for implementation on a VLSI chip, the speed of this multiplier is not competitive. There had been better designs proposed, that improved the speed of adding summands effectively [22] [23].

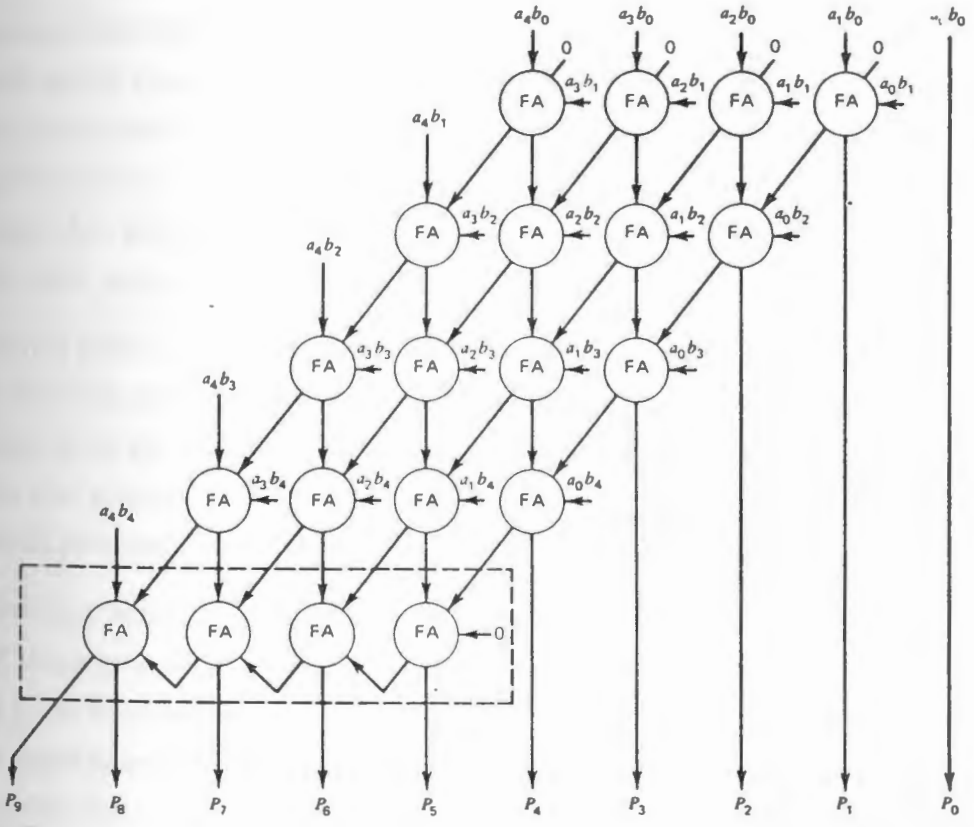


Figure 2.1: An array multiplier

2.3.2 Wallace Tree Reduction

The basic addition process employed in computers add two numbers together. The possibility exists of adding together more than two numbers in a single adder to produce a single sum. However, the logical complexity of the adder increases compared to the increase in resulting speed.

This method introduced addition of three numbers. This resulted in producing two numbers rather than a single number. These two numbers have different weights, and are represented by a sum and a carry. The advantage of this adder is, that it can operate without carry propagation along the digital stages. This form of adder is a simple full adder, where the carry inputs are used for the third input number, and the carry outputs for the second output number.

Wallace [13] introduced a tree structure which forms an interconnection of carry-save adders that reduces n partial products to two operands. The principle of the carry save adder is to use a single carry-save adder to reduce three bits of equal weight to two bits, one a sum and other a carry. Several adders have to be provided to reduce the partial products to two.

The maximum height of a column is n bits and it can be divided into three bit groups. Each of these groups can be reduced simultaneously to two bits, resulting in a new column $\frac{2}{3}n$ in height. The new column is again divided into three-bit groups, and the process repeats until the final column height is two bits. The time required varies linearly with the number of summands. The delay to reduce the height of a Wallace tree is $log_{3/2}(N/2)$. The base is derived from the fact that in full adder arrays, three partial products can be compressed to two at a time. Figure 2.2 shows a 7 bit-slice inputs that use pseudo-adders and are reduced to 3-bit slice.

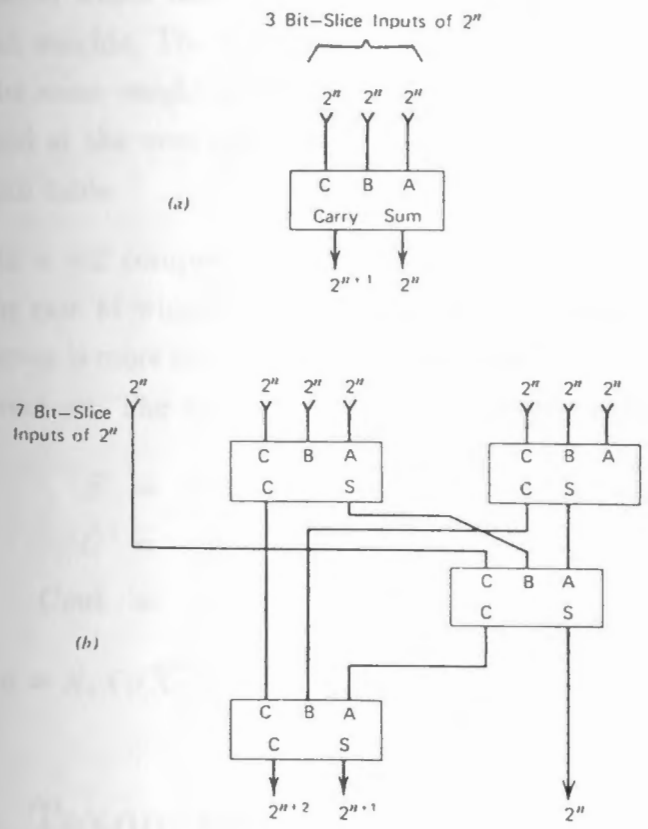


Figure 2.2: Wallace Tree

2.3.3 4:2 Compressor

This new unit adder was developed to enhance parallelism [24] and speed of the array part during circuit design [15]. It can compress four partial products of the same weight into two new partial products. Figure 2.3 is a block diagram for 4:2 adder. This 4:2 compressor has five inputs and three outputs. It is different from 5:3 counter which takes in 5 inputs of the same weight and produces 3 outputs of different weights. The sum output of 4:2 has weight 1 while the carry and C_{out} both have the same weight 2. For the four inputs taken in at one level, two outputs are produced at the next lower level. The 4:2 adder can be directly implemented from the truth table.

Since in a 4:2 compressor the four partial products are concurrently reduced to two, the rate at which the partial products are reduced is $log_2(N/2)$. Although 4:2 compressor is more complex, but is faster than Wallace tree and yields a more regular tree structure. The equations that represent the outputs of the 4:2 compressor are

$$S = \bar{C}in(X_1 \oplus X_2 \oplus X_3 \oplus X_4) + Cin(\overline{X_1 \oplus X_2 \oplus X_3 \oplus X_4}) \quad (2.11)$$

$$C = (X_1 + X_2)(X_3 + X_4) \quad (2.12)$$

$$Cout = \alpha + Cin(X_1 \oplus X_2 \oplus X_3 \oplus X_4), \quad (2.13)$$

where $\alpha = X_1X_2(\overline{X_3 \oplus X_4}) + \bar{X}_1\bar{X}_2X_3X_4$.

2.4 Taxonomy of multipliers

Different kinds of multipliers were studied between the period beginning early 1980's till 1991. Table 2.5 and 2.6 categorizes multipliers based on the recoding technique, generation of partial product, speed, size and technology. The speed of the multiplier improved exponentially with the enhancement in the technology. It can be observed from the table that the fastest multiplier does 53-bit multiplication in less than 10ns.

Table 2.5: Taxonomy of Multipliers

Source	Area	Speed	Recoding	Reduction
[25]	1.5×0.4 mm×mm	-	-	Carry Save Array
[26]	0.61×0.58 mm×mm	9.5ns	Array	Carry save adder
[27]	- mm×mm	7.5ns	Modified Booth	Modified array
[20]	5.8×6.3 mm×mm	120 ns	Modified Both	Redundant binary addition
[15]	3.8×6.5 mm×mm	120 ns	Booth's	4:2
[28]	1.55×1.44 mm×mm	6.75ns	Modified Booth	Wallace R-algorithm
[29]	- mm×mm	-	Non-Redundant radix 4	
[24]	2.68×2.71 mm×mm	15ns	Modified Booth	4:2
[24]	1.3×3.1 mm×mm	3.8ns	-	Wallace tree
[19]	-	22ns	5 bit recoding	
[19]	-	29ns	3 bit recoding	
[30]	3.62×3.45 mm×mm	10 ns	Modified Booth	4:2

Table 2.6: Taxonomy of Multipliers contd.

Source	Area	Summand	Technology	Length
[25]	1.5×0.4 mm×mm	-	NMOS pipeline	8×8
[26]	0.61×0.58 mm×mm	-	-	8×8
[27]	-	Carry propagate	0.6μm CMOS	16×16
[20]	5.8×6.3 mm×mm	CLA.	L.S.I nE/D MOS 2.7μm	8×8
[15]	3.8×6.5 mm×mm	Carry propagate	1.6μm CMOS	64×64 pipelined
[28]	1.55×1.44 mm×mm	CLA	.5μm CMOS	16×16
[31]	-	-	3μm CMOS	16×16
[24]	2.68×2.71 mm×mm	Carry select adder	.8μm CMOS	32×32
[24]	1.3×3.1 mm×mm	CLA	.8μm pass trans. logic	16×16
[19]	-	CSA array	-	16×16
[19]	-	Carry Propagate	-	16×16
[30]	3.62×3.45 mm×mm	Carry Select Adder	.5μm CMOS	54×54

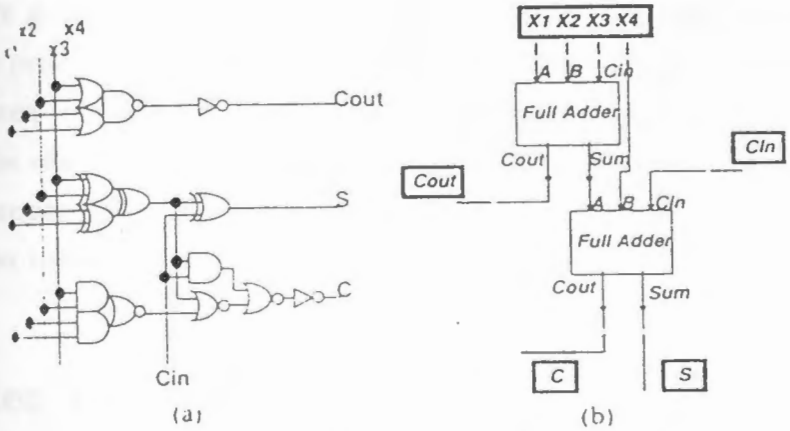


Figure 2.3: 4:2 Compressor

Chapter 3

Design and Area Evaluation

The area time performance is the most important governing factor in VLSI devices as well as any multiplier designs [32]. The variety of existing designs are due to the effort to compromise between area and speed. These designs vary in the method the partial products are generated and the method these partial products are reduced to only two partial products. Confusion still exists when comparing two multiplier designs for their area-time performance. The set of design rules and the device characteristics of the chosen technology determine the area and speed of the circuit. Obviously, a fair comparison must be based on the microelectronic architecture level to provide an unbias basis.

3.1 Recoder Design

Recoder is one of the most important unit in a fast multiplier. It performs one of the basic function of reducing the summands of the multiplier. Since the area occupied by the recoder has to be minimum, the design selected should not defeat the purpose of chosing the method of recoding to generate the partial product. The gate delay governs the speed of the recoder. The design should have a minimum delay for a fast multiplier. Designs of 3-bit and 5-bit recoder are discussed in this section.

3.1.1 3-bit recoder

The design of a 3-bit recoder was suggested by Ghest [8]. This is easily implementable. Different logic gates are used to make it area and time efficient. Figure 3.1 shows the design of a multiplier using a 3 bit recoder. The multiplier and the multiplicands are assumed to be of n-bits in length. Each encoder has three inputs coming from the multiplier Y. The output of the encoder generates three control signals to select one of the multiples of X: $(0, \pm X, \pm 2X)$. These three control signals are:

$$m_2 = Y_{n-1} \oplus Y_n \tag{3.1}$$

$$m_1 = Y_{n-2} \tag{3.2}$$

$$m_0 = Y_{n-2}Y_{n-1}Y_n + \bar{Y}_{n-2}\bar{Y}_{n-1}\bar{Y}_n \tag{3.3}$$

These control signals are given to 2:1 multiplexers. There are $n+1$ multiplexers for each row. The input to the multiplexers is different bits of the multiplicand. The multiplexer control signals select one of the two inputs. The different combination for selecting the signals are as follows:

$$\begin{aligned} m_2 &= 1 \quad X \\ &= 0 \quad 2X \end{aligned} \tag{3.4}$$

$$\begin{aligned} m_1 &= 0 \quad X \quad or \quad 2X \\ &= 1 \quad \bar{X} \quad or \quad 2\bar{X} \end{aligned} \tag{3.5}$$

$$\begin{aligned} m_0 &= 0 \quad \text{do not clear all multiplier bits} \\ &= 1 \quad \text{clear all multiplier bits} \end{aligned}$$

The output of the multiplexers forms *one* bit of the CSA array. The number of 3-bit Booth encoder required are $n/2$ for a n-bit multiplier.

3.1.2 5-bit recoder

Since 5-bit recoder technique is not accepted universally, the 5-bit recoder was designed to meet the requirements. Although [19], a multiplier using 5-bit recoding

was designed but the design itself was not discussed. Figure 3.2 shows the design of a multiplier using a 5-bit recoder. There are $n/4$ 5-bit encoders each having *five* inputs of the multiplier X and generating the *four* control signals for the multiplexers. These control signals are derived in Table 3.1 and are represented by S_4, S_3, S_2 and S_1.

The following equations, represents the four signals and the sign which is controlled by k, are derived from Karnaugh map.

$$S_0 = X_0X_{-1} + X_{-1}\bar{X}_{-2} + \bar{X}_{-3}X_{-1} + X_{-2}X_{-3}\bar{X}_{-1} \quad (3.6)$$

$$S_1 = X_{-1}X_{-2}X_{-3} + \bar{X}_{-2}X_{-3}X_0 + X_{-2}\bar{X}_{-3}X_0 + X_0\bar{X}_{-1}\bar{X}_{-3} \quad (3.7)$$

$$S_2 = X_{-2} \oplus X_{-3} \quad (3.8)$$

$$S_3 = X_0X_{-1}\bar{X}_{-2}X_{-3} + X_0\bar{X}_{-1}X_{-2}X_{-3} \quad (3.9)$$

$$k = X_0X_{-1}(X_{-2} \oplus X_{-3}) \quad (3.10)$$

(3.11)

There are two 4:1 multiplexers and one full adder that is required to generate each bit. There are $2(n+3)$ multiplexers in each row. These multiplexers select different multiples of the multiplicand Y. The sum output of the adder generates one of the possible multiples of Y $(0, \pm 1Y, \cdots, \pm 8Y)$. The k input to the adder controls the sign of the output. Thus the output from each adder forms an array, that is reduced by different partial product reduction techniques.

3.2 Area Evaluation

The area factor is an important factor when the economy of the design is a concern. The area could affect the size of a chip by a considerable amount. Here, we consider two partial product generation approaches: the 3 bit recoding (or modified Booth recoding) [11] and the 5 bit recoding [19]. The reduction of the partial products is done by using array of full adders, Wallace tree [13], or 4:2 compressors [24].

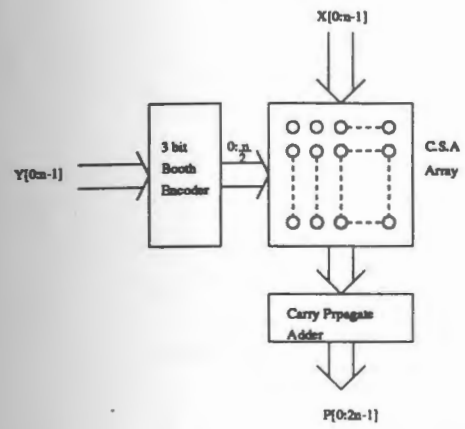


Figure 3.1: Design of a multiplier with a 3-bit recoder

Table 3.1: Generation of control signals for the multiplexers

$X_0X_{-1}X_{-2}X_{-3}$		$Y_3Y_2Y_1Y_0$	$S_3S_2S_1S_0$
0000	0	0000	0000
0001	+1	Y_3000	0100
0010	+1	Y_3000	0100
0011	+2	$0Y_200$	0001
0100	+2	$0Y_200$	0001
0101	+3	Y_3Y_200	0101
0110	+3	Y_3Y_200	0101
0111	+4	$00Y_10$	0010
1000	+4	$00Y_10$	0010
1001	+5	Y_30Y_10	0110
1010	+5	Y_30Y_10	0110
1011	+6	$0Y_2Y_10$	1001
1100	+6	$0Y_2Y_10$	1001
1101	+7	$\bar{Y}_300Y_0$	0111
1110	+7	$\bar{Y}_300Y_0$	0111
1111	+8	$000Y_0$	0011

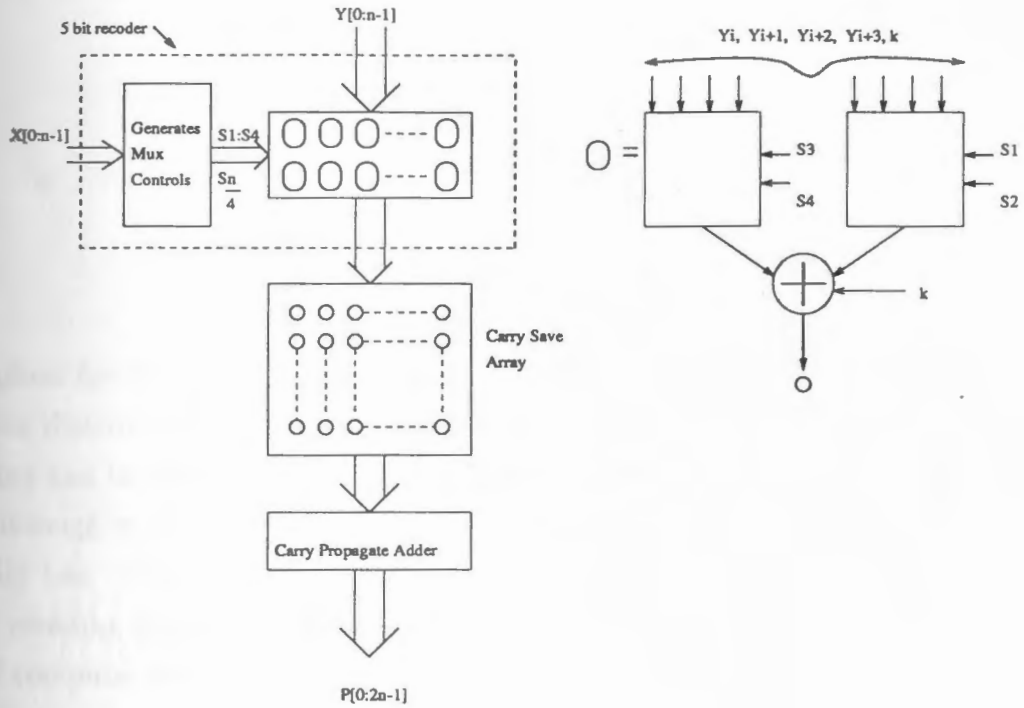


Figure 3.2: Design of a multiplier with a 5-bit recoder

3.2.1 Notations

In the area evaluation, the area of a full adder is used as a unit of measurement. The area of a full adder: a_f, is approximately double the size of a half adder: a_h, and the area of a 4:2 compressor a_c is δ units. We further assume the area of a 3 bit recoder: a_{r3}, equals α units. The area of a 5 bit recoder: a_{r5}, is assumed to be equal to $\alpha\beta$ units. Here β is the ratio of the areas of the 5 bit recoder and that of the 3 bit recoder. The above assumptions can be represented in the following mathematical form.

$$a_h = 0.5a_f \tag{3.12}$$

$$a_c = \delta a_f \tag{3.13}$$

$$a_{r3} = \alpha a_f \tag{3.14}$$

$$a_{r5} = \beta a_{r3} = \alpha\beta a_f \tag{3.15}$$

The multipliers for 8, 12, 16, 24 and 32 bits are examined respectively. The partial products are distributively generated using 3-bit and 5-bit recoding, respectively. The recoding can be distributive, or by shifting the multiplier bits in a centralized device. Although more area is needed when using the former, the time delay is substantially less. Hence there is a tradeoff between area and time. Distributive method of recoding is used for the analysis purpose as speed is the prime factor in high-speed computer arithmetic.

3.2.2 Multiplier Recoding

For a 3-bit recoded multiplier, $\frac{n}{2}$ recoders are required to generate all the partial products simultaneously. The generated partial products are $n+1$-bit in length. For each row, a 3-bit recoder and n multiplexers are needed. Let the area of a multiplexer and two additional gates be $\gamma_3 a_{r3}$, then the area used for recoding for each row is

$$a_{row3} = (1 + n\gamma_3)a_{r3} \tag{3.16}$$

For a 5-bit recoded multiplier, $\frac{n}{4}$ recoders are required to generate all the partial products simultaneously. Since the partial products are ranged from $8Y$ to -$8Y$, the recoded partial products are $n+3$-bit in length. In this case, $n+2$ multiplexers are needed. Note, that there are *two* four-to-one multiplexers used in 5-bit recoding. Let the area of these multiplexers and the full adder be $\gamma_5 a_{r5}$. Then the area occupied by the recoding circuits at each row is

$$a_{row5} = (1 + (n+2)\gamma_5)a_{r5} \tag{3.17}$$

3.2.3 Partial Products Reduction

Array of Full/Half Adders

Once the partial products are obtained, they are added using a known partial products reduction technique. An array of full adders such as in the Braun's multiplier [21] is a simple and straightforward approach. In this type of approach partial products are generated without recoding. We assume recoding for generation of partial products. The array of bits is assumed to be uniform. In the array multiplier half adders and full adders are used to form the final product. The carry is rippled through different stages. The final adder stage is eliminated in this multiplier. In order to do a fair comparison between different kinds of multiplier, the evaluation of area and speed is done on the array with one less summand. This reduces the height of the array by one row. Thus the result that the new array generates, together with the last summand forms the final stage of multiplication. This method makes the array multiplier comparable with other types of multipliers having similar features.

The number of half adders and full adders in the array is a function of n. They can be represented by $H_3(n)$ and $F_3(n)$ respectively, where n is the bit length of the multiplier and the multiplicand. For a multiplier with 3-bit recoding, the size of the array formed is $(n+1) \times (\frac{n}{2} - 1)$ bits. The number of half adders and the number

of full adders needed to reduce this array size is computed to be

$$H_3(n) = \frac{3n-6}{2} \tag{3.18}$$

and

$$F_3(n) = \frac{1}{4}(3n-6)(n-6) \tag{3.19}$$

respectively.

The total area required for this modified array multiplier with a 3-bit recoding is calculated from the following equation

$$A_{am3} = a_{r3} \times \frac{n}{2}(1+n\gamma_3) + a_f F_3(n) + 0.5 a_f H_3(n) \tag{3.20}$$

Substituting the above derived relations the area is computed to be

$$A_{am3} = \alpha \times \frac{n}{2}(1+n\gamma_3)a_f + \frac{1}{4}(3n-6)(n-6) + 0.5\frac{(3n-6)}{2} \tag{3.21}$$

The size of the array is $(n+3) \times (\frac{n}{4}-1)$ if 5-bit recoding is done. The number of half adders required is

$$H_5(n) = \frac{7n-28}{4} \tag{3.22}$$

and the number of full adders needed are

$$F_5(n) = \frac{1}{4}(7n-28)(n-8). \tag{3.23}$$

The total area needed for the adder is calculated from the equation given below:

$$A_{am5} = a_{r5} \times \frac{n}{4}(1+(n+2)\gamma_5) + a_f F_5(n) + 0.5 a_f H_5(n) \tag{3.24}$$

After substituting the values for $H_5(n)$ and $F_5(n)$ the area is computed to be

$$A_{am5} = \alpha\beta a_f \times \frac{n}{4}(1+(n+2)\gamma_5) + \frac{1}{4}(7n-28)(n-8) + 0.5\frac{7n-28}{4}. \tag{3.25}$$

Wallace Tree

Wallace tree consists of carry save adders (which are essentially full adders) to gain more time efficiency. In the case of 3-bit recoding, $F_{3w}(n)$ full adders and $H_{3w}(n)$ half adders are required. These two functions cannot be formulated, as they are not deterministic function of n. The actual values has to be derived by actual implementations for different n. The values of $F_{3w}(n), H_{3w}(n), F_{5w}(n)$ and $H_{5w}(n)$ used in the next section are those derived from actual implementation. The area occupied by Wallace tree in an n-bit multiplier using 3-bit recoding can be calculated by the following equation.

$$A_{wp3} = a_{r3} \times \frac{n}{2}(1 + n\gamma_3) + a_f F_{3w}(n) + 0.5 a_f H_{3w}(n) \tag{3.26}$$

Using the relationships from eq. 3.14, the above equation can be rewritten as follows.

$$A_{wp3} = \alpha \times \frac{n}{2}(1 + n\gamma_3) + F_{3w}(n) + 0.5 H_{3w}(n) \tag{3.27}$$

If 5-bit recoding is used, $F_{5w}(n)$ full adders and $H_{5w}(n)$ half adders are present in the Wallace tree. The area of the Wallace tree in an n-bit multiplier using 5-bit recoding is derived as follows.

$$A_{wp5} = a_{r5} \times \frac{n}{4}(1 + (n+2)\gamma_5) + a_f F_{5w}(n) + 0.5 a_f H_{5w}(n). \tag{3.28}$$

From (3.14) and (3.15),

$$A_{wp5} = \alpha\beta \times \frac{n}{4}(1 + (n+2)\gamma_5) + F_{5w}(n) + 0.5 H_{5w}(n). \tag{3.29}$$

4:2 Compressors

The 4-2 compressors are used to speed the array part during circuit design. For the sake of simplicity, the full adders are replaced by 4-2 compressors to reduce *four* partial products to *two*. The fourth input is fed with a zero. In the case of 3-bit recoding, $C_3(n)$ 4-2 compressors are required. The area occupied by 4-2 compressors in an n-bit multiplier can be calculated by the following equation.

$$A_{c3} = a_{r3} \times \frac{n}{2}(1 + n\gamma_3) + a_c C_3(n) + 0.5 H_{3c}(n) a_f \tag{3.30}$$

Using equation (3.14), the above equation can be rewritten as follows

$$A_{c3} = \alpha \times \frac{n}{2}(1 + n\gamma_3) + \delta C_3(n) + 0.5H_{3c}(n). \tag{3.31}$$

If 5-bit recoding is used, there are $C_5(n)$ 4-2 compressors. The area occupied by 4-2 compressors in an n-bit multiplier using 5-bit recoding is derived as follows.

$$A_{c5} = a_{r5} \times \frac{n}{2}(1 + n\gamma_5) + a_c C_5(n) + 0.5H_{5c}(n)a_f \tag{3.32}$$

Using equation (3.13), (3.14) and (3.15)

$$A_{c5} = \alpha\beta \times \frac{n}{2}(1 + n\gamma_5) + \delta C_5(n) + 0.5H_{5c}(n) \tag{3.33}$$

Again $C_3(n), H_{3c}(n), C_5(n)$ and $H_{5c}(n)$ cannot be formulated systematically. The values used in the next section for the purpose of comparison are based on actual implementations.

3.2.4 Area Comparisons

The areas of the multipliers with 3-bit recoding and 5-bit recoding are compared in this section, for different techniques of partial product reduction. These areas give the cost estimation of the multipliers.

Array of Full/Half Adders

The reduction of partial products in the multiplier is a deterministic function of n. Therefore the areas cannot be solely represented by the full adder units. The evaluation of array multipliers is done in order to give a complete treatment to the multipliers.

Wallace Tree

The areas required for n bit multiplication, using 3-bit and 5-bit recoding are equated to be equal. Then the value of β can be calculated for a given value of α. The value

of β obtained gives the ratio of a 5-bit to 3-bit recoder, that is required, in order to have the same size of the multiplier.

Equating A_{wp3} and A_{wp5}, we obtain

$$\beta = \frac{[\alpha \times \frac{n}{2}(1 + n\gamma_3) + F_{3w}(n) - F_{5w}(n) + 0.5(H_{3w}(n) - H_{5w}(n))]}{\alpha \times (1 + (n+2)\gamma_5)\frac{n}{4}} . \qquad (3.34)$$

The results are plotted for different values of n against the value of β. We assume that γ_3=0.4 and γ_5=0.25 based on the finding in the trial VLSI layouts. Figures 3.3 and 3.4 show the VLSI layouts by MAGIC of a 3-bit and 5-bit recoders respectively. We assume a 2μm p-well CMOS process. The value of α is varied from 3 to 8. Table 3.2 lists the values of $F_{3w}(n)$ and $H_{3w}(n)$ for different values of n. Table 3.3 shows the values of $F_{5w}(n)$ and $H_{5w}(n)$.

In other words, with the given parameters, if the area of a 5-bit recoder is less than four times that of a 3-bit recoder, the multiplier with 5-bit recoding is more efficient in area. Figure 3.5 shows several plots based on the above equations. Note that the plots represent the upper bounds of β for the given values of α and n. For example, when n=16 and α=5, we derive that β=4.4. This means that when a 3-bit recoder is five times the size of a full adder, for n=16, a 5-bit recoding multiplier using Wallace tree is more area efficient if the size of a 5-bit recoder is less than 4.4 times the size of a 3-bit recoder.

We emphasize that this bound is based on two fundamental variables: the size of the 3-bit recoder and the size of the 5-bit recoder. The value of α must br based on the most compact layout of the 3-bit recoder and the full adder.

4:2 Compressors

The areas required for n-bit multiplication using 3-bit and 5-bit recoding are equated to be equal. The value of β is calculated for different values of α, as it was done for

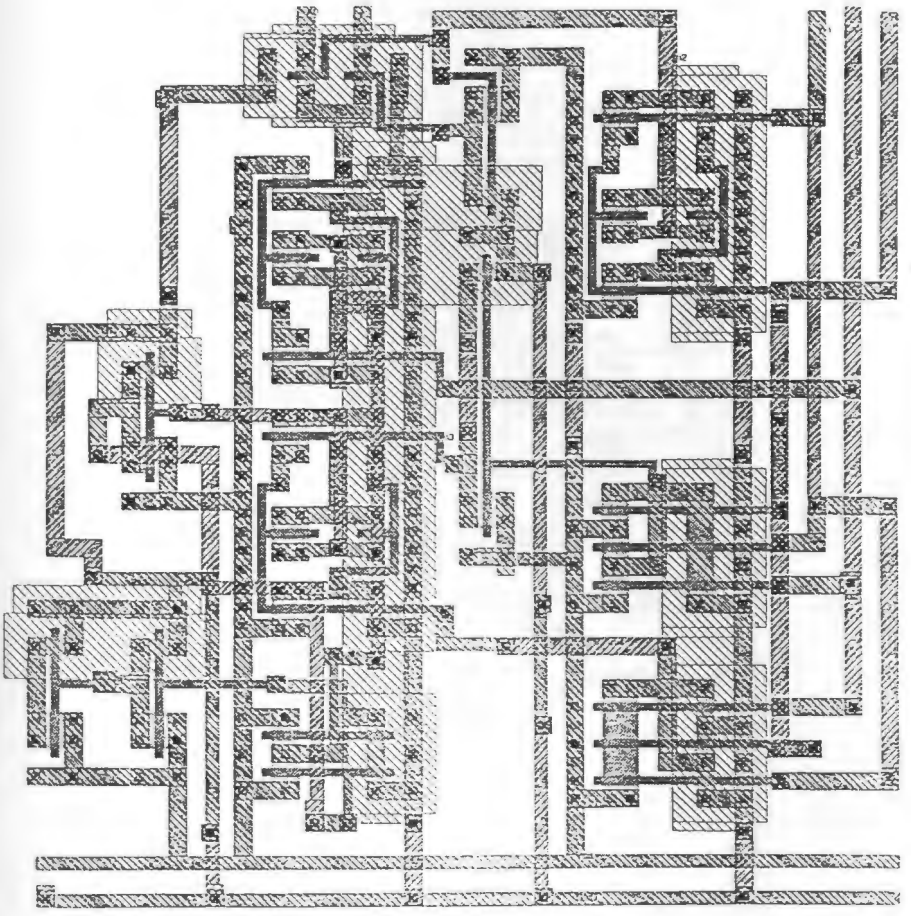


Figure 3.3: MAGIC Layout of a 3-bit recoder

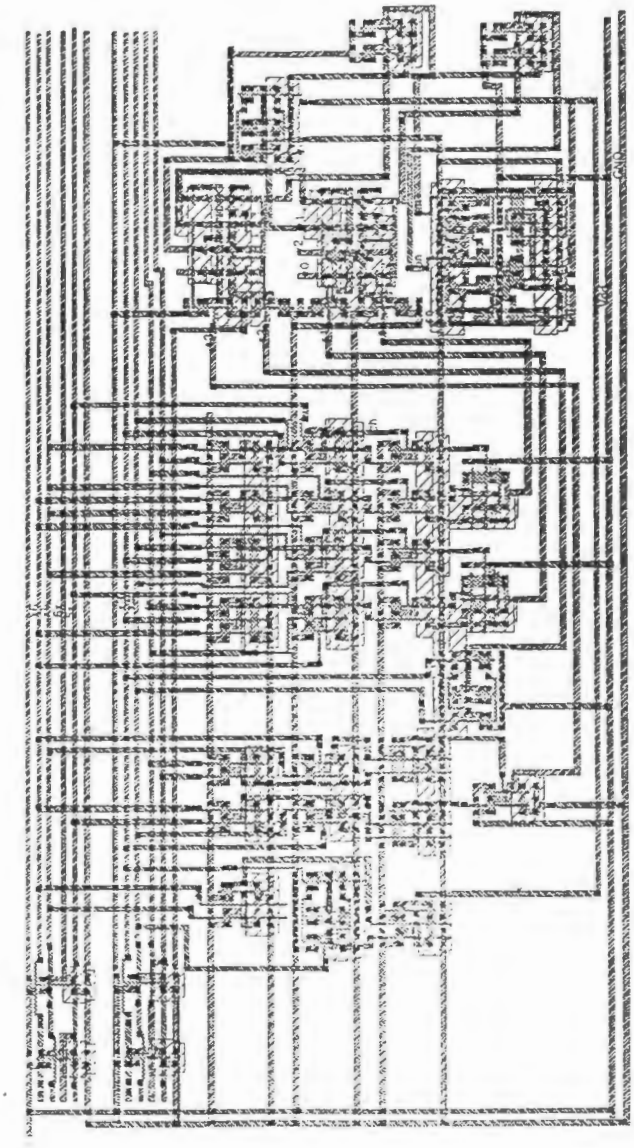


Figure 3.4: MAGIC Layout of a 5-bit recoder

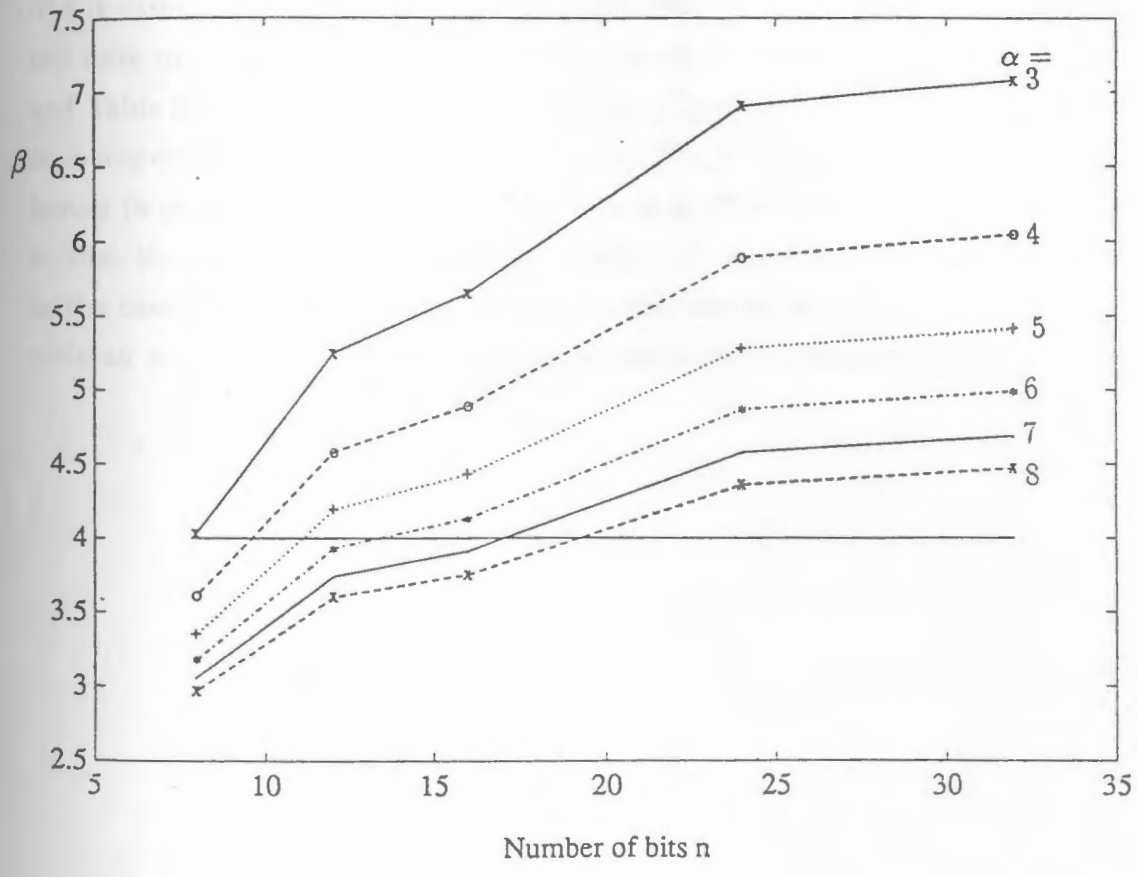


Figure 3.5: n vs β for a Wallace tree

the Wallace tree. Equating A_{c3} and A_{c5}, we obtain

$$\beta = \frac{\alpha \times \frac{n}{2}(1 + n\gamma_3) + \delta(C_3(n) - C_5(n)) + 0.5(H_3(n) - H_5(n)}{\alpha \times (1 + (n+2)\gamma_5)\frac{n}{4}} . \tag{3.35}$$

The results are plotted for different values of n against the value of β. Here, δ is assumed to be 2, because 4:2 compressor is twice the size of a full adder. The value of δ is varied from 1.5, 2.0 and 2.5 to obtain different plots. The variation in δ does not have much affect on the plots. The value of α is varied from 3 to 6. Table 3.4 and Table 3.5 lists the values of $C_3(n), H_{3c}(n), C_5(n)$ and $H_{5c}(n)$ for different values of n respectively. Again the plots in Figures 3.6, 3.7 and 3.8 represents the upper bound in each case for a given value of n and α. One interesting observation here, is, that the upper bounds for multipliers with 4:2 compressors are much lower than in the case of Wallace tree multipliers. In other words, it is much more difficult to yield an area efficient 5-bit recoding multiplier when 4:2 compressors are used.

Table 3.2: Adder units required for Wallace Tree using 3-bit recoding

Number of bits (n)	Number of full adders $F_{3w}(n)$	Number of Half adders $H_{3w}(n)$
8	10	7
12	36	18
16	79	32
24	212	63
32	403	115

Table 3.3: Adder units required for Wallace Tree using 5-bit recoding

Number of bits (n)	Number of full adders $F_{5w}(n)$	Number of Half adders $H_{5w}(n)$
8	-	-
12	7	4
16	22	13
24	74	27
32	156	70

Table 3.4: Adder units required for 4:2 compressors using 3-bit recoding

Number of bits (n)	Number of 4:2 Compressors $C_3(n)$	Number of Half adders $H_{3c}(n)$
8	10	7
12	22	4
16	45	9
24	118	20
32	217	38

Table 3.5: Adder units required for 4:2 Compressors using 5-bit recoding

Number of bits (n)	Number of 4:2 Compressors $C_5(n)$	Number of Half adders $H_{5c}(n)$
8	-	-
12	7	4
16	15	4
24	46	8
32	88	22

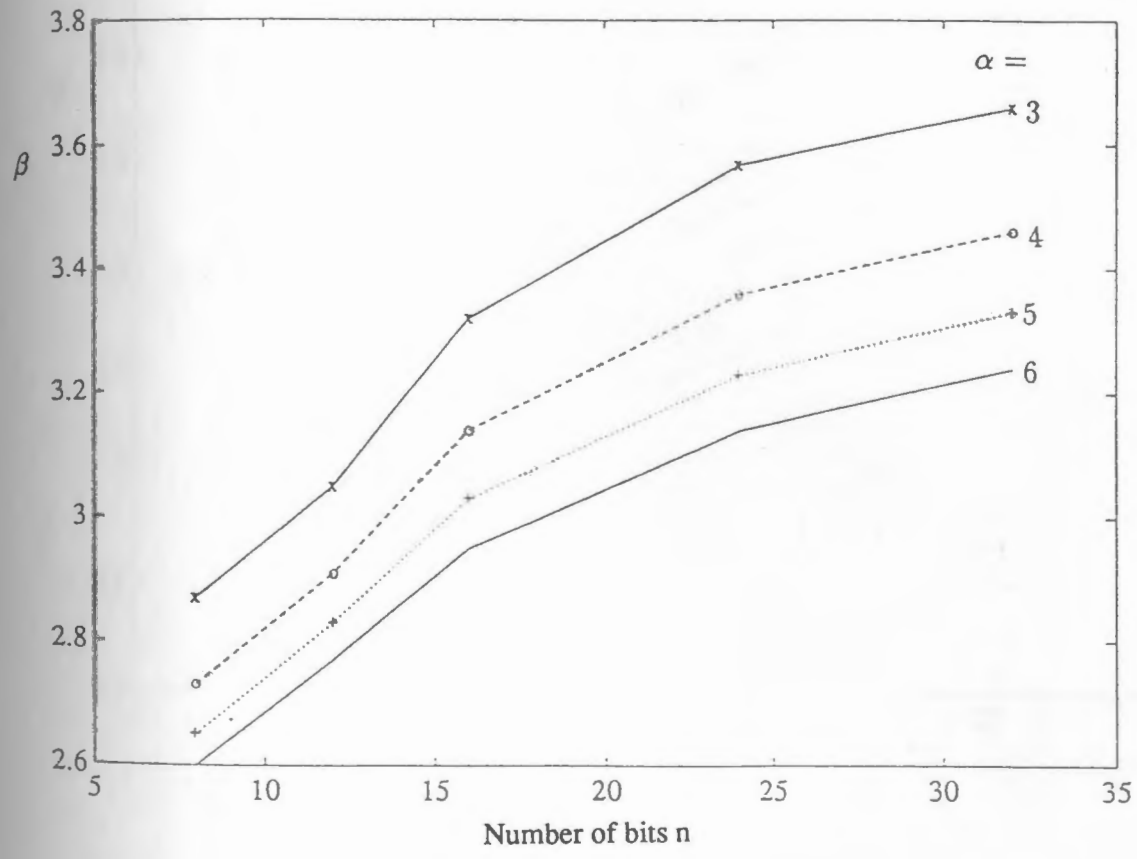


Figure 3.6: n vs β for a 4:2 compressor for δ= 1.5

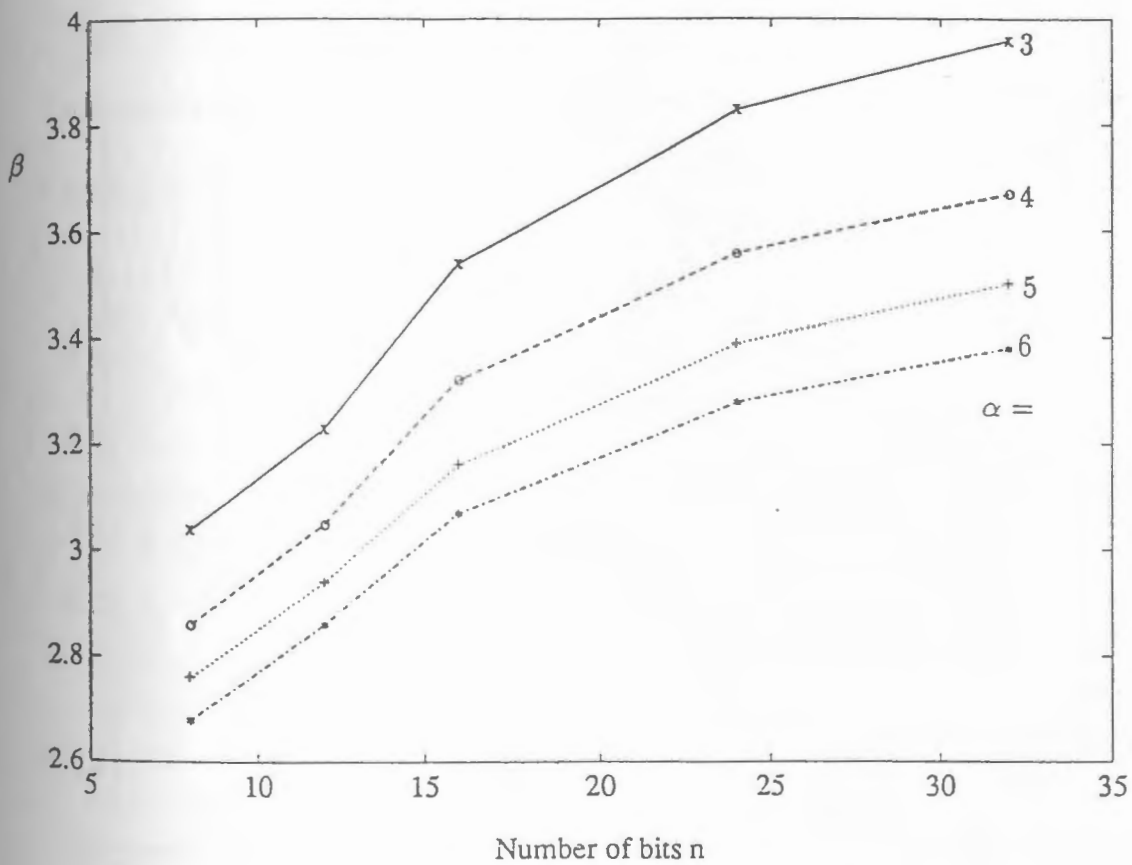


Figure 3.7: n vs β for a 4:2 compressor for δ= 2.0

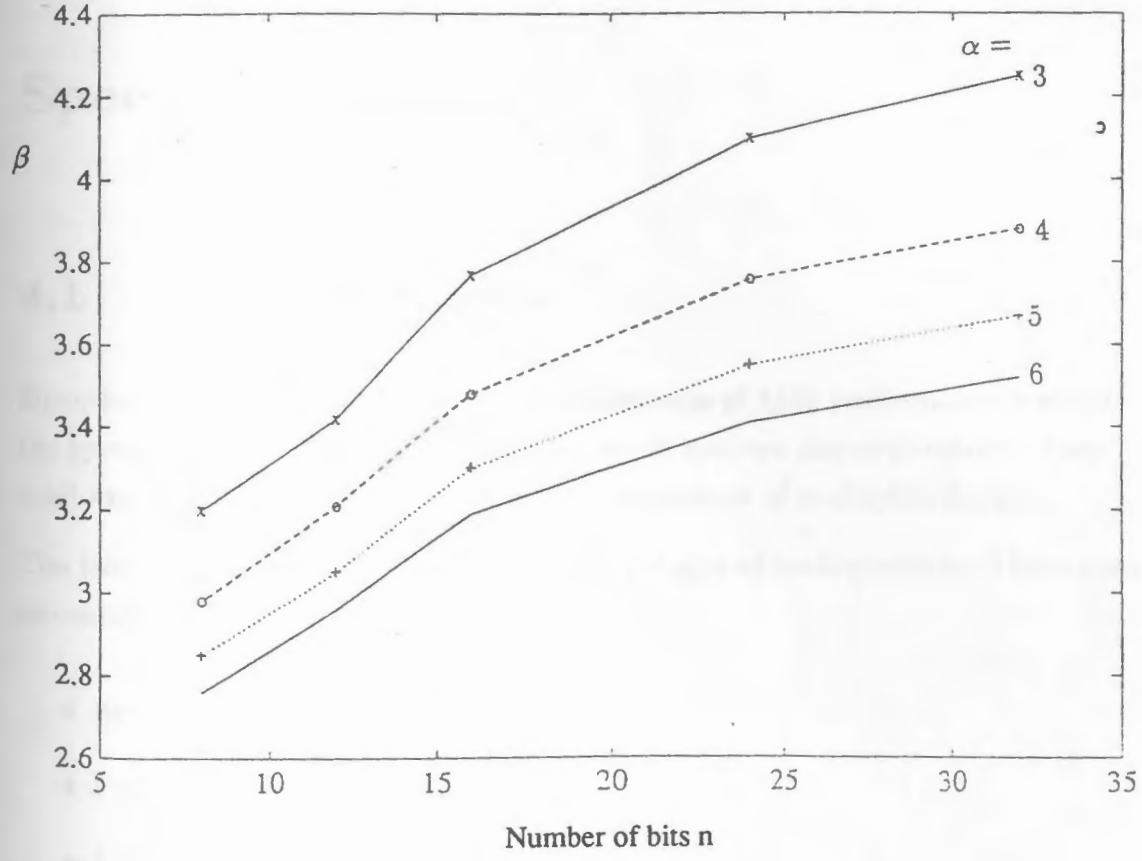


Figure 3.8: n vs β for a 4:2 compressor for δ= 2.5

Chapter 4

Speed Evaluations

4.1 Time Evaluations

Since fast arithmetic circuits are the key elements of high performance computers, the speed of the multiplier is a major concern in modern day applications. Here, we shall examine the delay times of various constructions of multiplier designs.

The timing analysis is carried out in different stages of multiplication. These stages are subdivided into three stages:

- Recoding stage
- Partial products reduction stage
- Final adder stage.

The delay time of recoding stage is the time taken for the encoder to do the recoding and the generation of partial product. The reduction of the partial products is dependent on the recoding method and the array organization. The final stage is a parallel adder with a delay of Δ_{cp}. This number is independent from the recoding method and the organization of partial products reduction.

4.1.1 Recoding Stage

The time taken for recoding can be represented by Δ_{3r} for a 3-bit recoder and Δ_{5r} for a 5-bit recoder. Since, a distributive approach is assumed for the generation of partial products the delay of the recoder is not a function of n. In [19] the recoding is assumed to be centralized. Then, the generation of odd products (3Y, 5Y, etc) needs $n - bit$ binary adders and the delay time is a function of n. This is is obviously not suitable for high-speed multiplication. This type of discussion will not be considered in the following discussions.

4.1.2 Partial Products Reduction Stage

The timing of an array multiplier is obtained by tracing the worst case critical path from which the partial products are generated to the final reduction of these products into a single row of final product. The delay in an array multiplier with a 3-bit recoder can be represented by the following equation.

$$\Delta_{3am} = \Delta_{3r} + [\frac{1}{4}(3n-6)(n-6)]\Delta_{FA} + 8\Delta + \Delta_{cp} \tag{4.1}$$

Here Δ is the time delay through one inverter. The term 8Δ represents the delay in the last row assuming that there is CLA adder, instead of allowing the carry to ripple through. The delay in the multiplier using a 5-bit recoder is:

$$\Delta_{5am} = \Delta_{5r} + [\frac{1}{4}(7n-28)(n-8)]\Delta_{FA} + 8\Delta + \Delta_{cp} \tag{4.2}$$

From (4.1) and (4.2) it is observed that simple array structure with 5-bit recoding is far better. However, it is trivial to see that this simple array structure is not a candidate for high-speed multiplication. The importance of 5-bit recoding must be justified by the state-of-the-art multiplier designs.

The number of carry save adder stages required to reduce the Wallace tree height from n to 2 is

$$S_w = \log_{1.5}(\frac{n}{4}). \tag{4.3}$$

The number of carry save adder stages required to reduce the partial products using 4:2 compressor is

$$S_c = \log_2(\frac{n}{4}). \tag{4.4}$$

The total delay in a 3-bit recoder and the partial product generation Δ_{3r} is the same for each row of partial product generation. Thus the total multiplication time needed for an n bit number using modified Booth recoding and Wallace tree is

$$\Delta_{3BW} = \Delta_{3r} + \log_{1.5}(\frac{n}{4})\Delta_{FA} + \Delta_{cp} \tag{4.5}$$

The total multiplication time needed for an n bit number using modified Booth recoding and 4:2 compressor is

$$\Delta_{3B4} = \Delta_{3r} + \log_2(\frac{n}{4})\Delta_{4,2} + \Delta_{cp} \tag{4.6}$$

The total delay in a 5-bit recoder and the partial product generation Δ_{5r} is the same for each row of partial product generation. The number of partial products generated in this case is $\frac{n}{4}$. Thus, the total multiplication time needed for an n bit number using 5-bit recoding and Wallace tree is

$$\Delta_{5BW} = \Delta_{5r} + \log_{1.5}(\frac{n}{8})\Delta_{FA} + \Delta_{cp}. \tag{4.7}$$

The difference in speed of the multiplier using a Wallace tree, with 3-bit and 5-bit recoding can be represented as

$$t_{wb} = \Delta_{3BW} - \Delta_{5BW} \tag{4.8}$$

$$= (\Delta_{3r} - \Delta_{5r}) + (log_{1.5}8 - log_{1.5}4)\Delta_{FA} \tag{4.9}$$

The total multiplication time needed for an n bit number using modified Booth recoding and 4:2 compressor is

$$\Delta_{5B4} = \Delta_{5r} + \log_2(\frac{n}{8})\Delta_{4,2} + \Delta_{cp}. \tag{4.10}$$

The 4:2 compressor can be realized using complex gates and this compressor is 1.5 times the speed of a full adder. Therefore the equations (4.6) and (4.10) can be rearranged to form

$$\Delta_{3B4} = \Delta_{3r} + 1.5\log_2(\frac{n}{4})\Delta FA + \Delta_{cp} \quad (4.11)$$

$$\Delta_{5B4} = \Delta_{5r} + 1.5\log_2(\frac{n}{8})\Delta FA + \Delta_{cp} \quad (4.12)$$

The difference in speed of the multiplier using a 4:2 compressor with 3-bit and 5-bit recoding can be represented as

$$t_{4b} = \Delta_{3B4} - \Delta_{5B4} \quad (4.13)$$

$$= (\Delta_{3r} - \Delta_{5r}) + 1.5\Delta_{FA} \quad (4.14)$$

The equations can be represented in natural logarithm as

$$\Delta_{3BW} = \Delta_{3r} + (2.5ln(n) - 1.71)\Delta_{FA} + \Delta_{cp} \quad (4.15)$$

$$\Delta_{3B4} = \Delta_{3r} + 1.5(3.32ln(n) - 1)\Delta_{FA} + \Delta_{cp} \quad (4.16)$$

$$\Delta_{5BW} = \Delta_{5r} + (2.5ln(n) - 3.42)\Delta_{FA} + \Delta_{cp} \quad (4.17)$$

$$\Delta_{5B4} = \Delta_{5r} + 1.5(3.32ln(n) - 2)\Delta_{FA} + \Delta_{cp} \quad (4.18)$$

For a multiplier of bit n, the time taken for a 3-bit and 5-bit recoder is dependent on the recoding technique used, and the way the partial products are generated. Since, recoding is done distributively the $\frac{n}{2}$ partial products and $\frac{n}{4}$ partial products for 3-bit and 5-bit respectively are generated simultaneously. The delay of partial product reduction with the change in the number of bits was studied for 3-bit and 5-bit recoders. The plots were obtained for *three* different methods of reduction namely 4:2 compressor(using two full adders) having a delay time of $2\Delta_{FA}$, Wallace Tree with a delay time of Δ_{FA} and 4:2 compressor (complex gates) having a time delay of $1.5\Delta_{FA}$. These were evaluated for both the recoding schemes. The plots are shown in Figures 4.1 and 4.2. It can be observed from these plots that the 4:2 compressor using complex gate is the fastest of the three techniques. It is interesting to point out that the evaluation in [17] uses 4:2 compressor with $2\Delta_{FA}$ delay time

and concludes that Wallace tree is better than 4:2 compressor using modified Booth recoding. The plots shown in Figures 4.1 and 4.2 indicate that such a conclusion is misleading.

4.2 Comparison

The speed comparisons of multipliers using different recoders for the Wallace tree and the 4:2 compressor is compared. The difference in speed stays constant for the two types of recoders, and is not a function of the number of bits. The different technologies for reducing partial products, also plays an important role in determining the speed of the multiplier. The 4:2 compressor that uses complex gate proved to be the fastest method for reducing the partial products. The total speed of the multiplier is determined by adding the recoder delay to the reduction technique. The 3-bit recoder is always faster than the 5-bit recoder. But the total delay is a function of the number of bits n. t_{wb} and t_{4b} are the upper bounds for the time efficiency of 5-bit recoding multipliers. For a positive t_{wb}, the 5-bit recoding multiplier in question has a shorter propagation time than its 3-bit recoding counterpart. If the derived t_{wb} is a negative number then the 5-bit recoding multiplier is not as time efficient. Similar principle applies to t_{4b}. Again we caution here that the delay time of the 3-bit recoder that is used to derive t_{wb} or t_{4b} must be based on the best possible design for a true comparison.

4.3 Empirical Results

The VLSI layout of the recoder is done using CAD tool MAGIC. The 3-bit recoder is designed according to [12]. The design of a 5-bit recoder is derived using truth tables. These designs are implemented using the standard gates. These gates are available in the cell library on the computers of the Electrical Engineering Department. In the trial layouts, the area of a 5-bit recoder is *four* times the area of a 3-bit recoder:

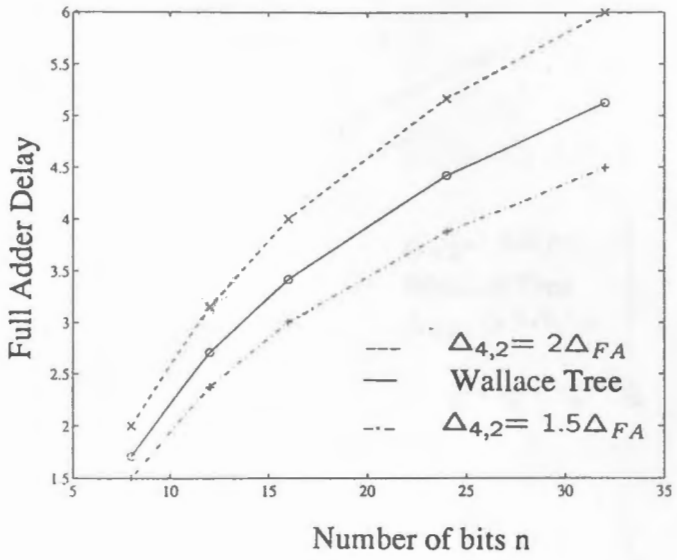


Figure 4.1: n vs Δ_{FA} for 3-bit recoding.

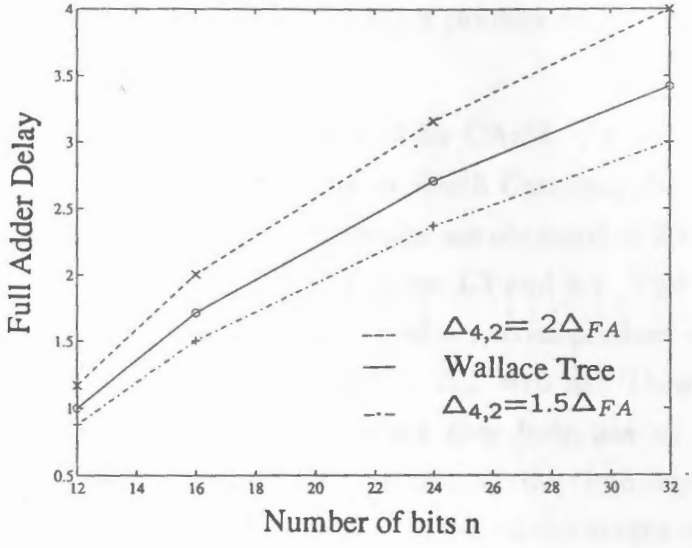


Figure 4.2: n vs Δ_{FA} for 5-bit recoding.

$\beta = 4$. The area of a 3-bit recoder is approximately *seven* times the area of a full adder: $\alpha = 7$. These areas are calculated using the approximation that the layout can be made more compact. In other words, most of the empty spaces in the layout was not counted. The layouts are shown in Figures 3.3 and 3.4 in the previous section. The actual areas are: for 3-bit recoders, 134428 μm^2 and for 5-bit recoder 573389 μm^2. Since these values can vary depending on the design chosen, therefore, there is a need to observe the results by varying the parameters. These results gives the practicality of a 5-bit recoder having a priority over a 3-bit recoder in a certain size of the multiplier.

The delay of the recoders are simulated by CAzM. CAzM is the circuit simulator available from Microelectronic Center of North Carolina(MCNC) and is installed in the Department's Sun 4 server. The results are obtained at 20MHz clock rate. These results are plotted using Sigview in Figures 4.3 and 4.4. The total time taken for 1 bit to propagate through the formation of a partial product Δ_{3r} is 8ns. The delay in propagation if 5-bit recoding is used is Δ_{5r} =10 ns. These values are measured between the 50% of the V_{DD}, when input rises from low to high, and that of the output signals. These values give approximately the time required for each stage of the partial product to form. The total time for all the stages equals the time for one stage.

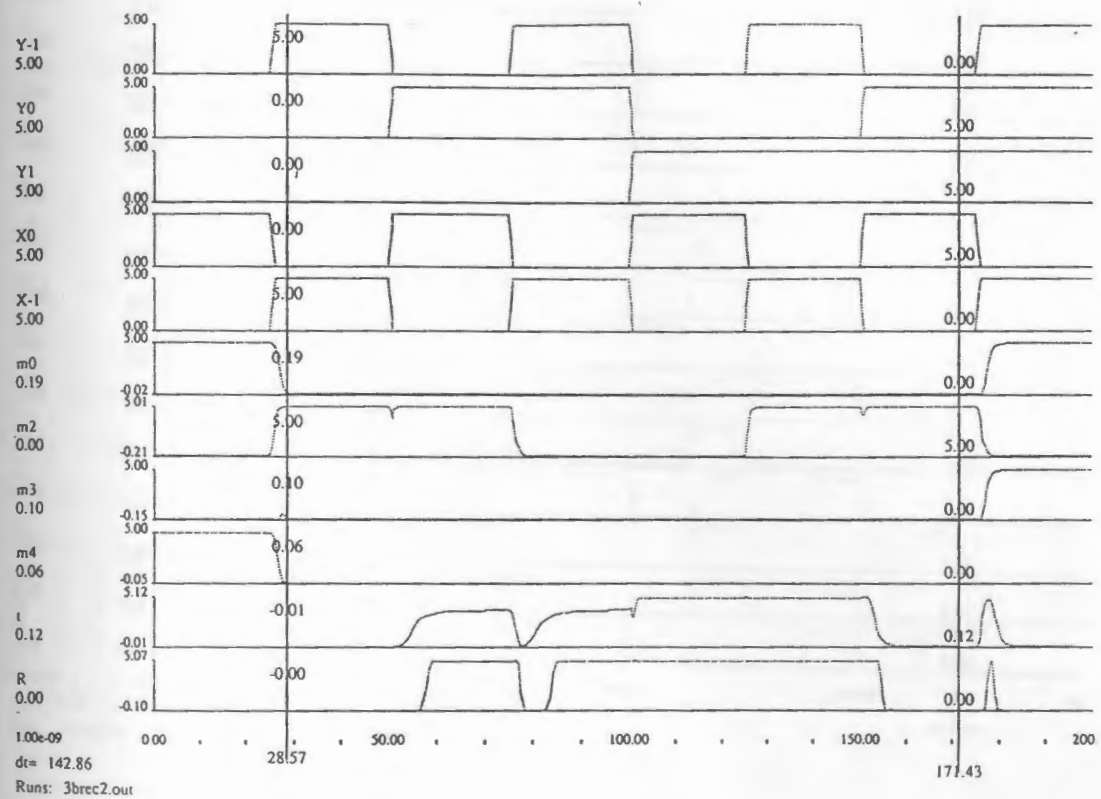


Figure 4.3: Timing diagram for a 3-bit recoder

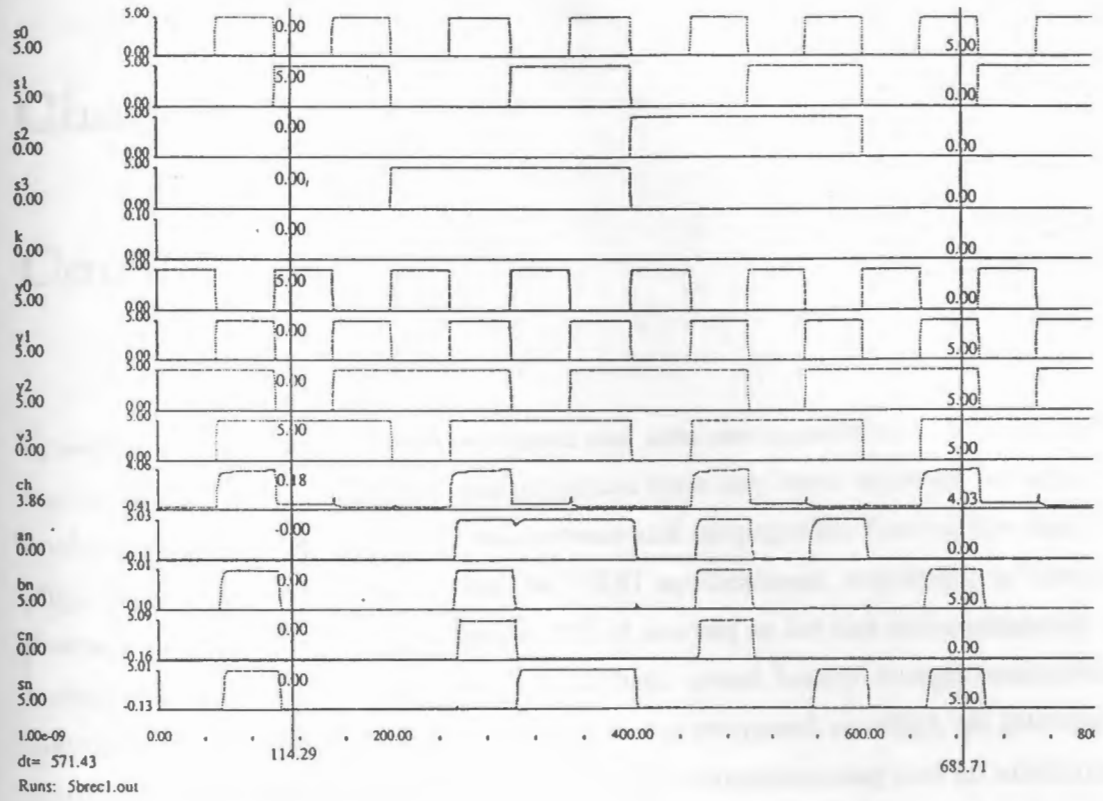


Figure 4.4: Timing diagram for a 5-bit recoder

Chapter 5

Conclusion

In recent studies, trade-off between speed and area are considered to be bottleneck in the multiplier design. A 10 ns multiplication time has been achieved by optimizing both the propagation time of 4:2 compressor and propagation time of the final adder [30]. It has also been known that for VLSI applications, regularity of the signal flow and the layout is very important [28]; it is even so for the submicrometer technology where the devices are fast. In [33] high speed binary integer multiplication algorithm suitable for VLSI implementation was proposed in which all intermediate results are represented in the redundant binary representations; and all additions are performed in the redundant binary number system, where addition of two numbers can be performed in a constant time independent of the word length of operands without carry-propagation. Multibit overlapped scanning technique can be applied to the multiplication function and possible actions to be taken for the design of the multipliers has been discussed [34].

In this study, we examine the microelectronic architectures of array multipliers (or hardware algorithms). The study is important in pointing out the unbiased approach for multiplier designs comparison. The speed-up contributed by technological innovation (reduction of feature size) and by fine-tuning at the switch or FET level is limited by its fundamental microelectronic architecture. In other words it is much more important to understand the merit of an array multiplier before committing to

it. Design of the 3-bit and the 5-bit recoders is implemented in VLSI for concrete numbers to be used in the study. The area of the recoders is calculated from the layouts drawn in MAGIC. For the comparisons of areas, appropriate ratios were assigned to full adders and the recoders. The number of full adders were calculated using the Wallace tree and the areas for 8, 12, 16, 24 and 32 bit multipliers were evaluated.

As far as the area is concerned, 5-bit recoding is not efficient for the multipliers of size less than 16-bits given $\alpha = 7$. In other words area efficient 5-bit recoder can be achieved for higher word lengths and an efficient 5-bit recoder design or layout. We caution that this conclusion was drawn without the consideration of the areas required for the routing channels. Nonetheless, the comparison given here represents the best approximation so far.

For the timing analysis, a 4:2 compressor using a complex gate gives the best performance for both 3-bit and 5-bit recoders. The recoder delays were not considered along with the delays for partial product reduction. When these delays are added to the partial product reduction delays, the total delay for the two stages can be more accurately modeled. Thus, the total timings also depends upon the recoding techniques being used. In general, the total time required for using 5-bit recoder is more than that required using a 3-bit recoder.

When using 4:2 compressors, a 5-bit recoding multiplier may not be efficient in area for more than 32 input bits. Since, the evaluations for higher than 32-bits were not performed here, the above statement is based on the trend shown in the plots. In this configuration 5-bit recoding multipliers are always faster than their 3-bit recoding counterparts. The second set of evaluation is done on a 4:2 compressor. The size of the multiplier should be more than 32 bits to have an area efficient use of a 5-bit recoder. Note that, this may be true only if the given design for recoders is used.

In the empirical result, we have demonstrated, the approximate speed of the 3-bit recoder and the 5-bit recoder. The 3-bit recoder design is apparently faster than the 5-bit recoder, but the ratio between them may vary depending upon the circuit

design, layout and fabrication. In other words there is no "optimal" microelectronic architecture, or design, or chip. The only thing we can claim is, that under a given fabrication environment, the microelectronic architecture that is the best can be chosen.

References

[1] D. Hutchison and P. Silvester, *Computer Logic Principles and Technology*, pp. 173–190. England: Ellis Harwood Limited, 1987.

[2] R. F. Shaw, "Arithmetic operations in a binary computer," *Rev. Science Instrumentation*, 1950.

[3] S. Waser and M. J. Flynn, *Introduction to Arithmeic for Digital System Designers*, pp. 156–157. New York: CBS Publishing, 1982.

[4] J. B. Gosling, "Design of a large high-speed floating-point-arithmetic unit," *IEE Proceedings*, 1971.

[5] F. S. Anderson, J. G. Earle, R. E. Goldschmidt, and D. Powers, "The IBM system 360/91 floating point execution unit," *IBM Jounal*, vol. 11, pp. 34–53, Jan 1967.

[6] P. Brumm and D.Brumm, *80386 A Programming and Design Handbook*, pp. 434–443. PA: TAB Books Inc., 1989.

[7] A. D. Booth, "A signed binary multiplication technique," *Qt. J. Mech. Appl. Math.*, vol. 4, pp. 236–240, 1951.

[8] R. C. Ghest, "A two's complement digital multiplier," Technical report, Advanced Micro Devices, Sunnyvale, CA, Nov 1971.

[9] D. A. Pucknell and K. Eshraghian, *Principles of CMOS VLSI Design*, pp. 10–25. NJ: Prentice-Hall, 1988.

[10] A. Habibi and P. A. Wintz, "Fast multipliers," *IEEE Transactions on Computers*, pp. 153–157, Feb 1970.

[11] O. L. MacSorley, "High speed arithmetic in binary computers," in *Proceedings IRE*, Jan 1961.

[12] K. Hwang, *Computer Arithmetic Principles, Architecture and Design*, ch. 6. New York: Wiley, 1979.

[13] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Transactions on Computers*, vol. EC-13, Feb 1964.

[14] E. E. Swartzlander, *Computer Arithmetic*, pp. 118–125. Vol. 1, Los Alamitos, CA: IEEE Computer Society Press, 1987.

[15] M. R. Santoro and M. A. Horowitz, "A Pipelined 64 × 64-bit Iterative Multiplier," *IEEE Journal of Solid-State Circuits*, vol. 24, pp. 487–493, April 1989.

[16] O. J. Bedrij, "Carry select adders," *IRE Transactions Electronics Computers*, vol. EC-11, pp. 65–73, June 1962.

[17] P. J. Song and G. D. Michelli, "Circuits and Architecture Trade-Offs for High-Speed Multiplication," *IEEE Journal of Solid-State Circuits*, vol. 26, pp. 1184–1192, Sept 1991.

[18] L. P. Rubinfield, "A proof of the modified Booth's algorithm for multiplication," *IEEE Transactions on Computers*, Oct 1975.

[19] H. Sam and A. Gupta, "A Generalized Multibit Recoding of two's Complement Binary Numbers and its Proof with Application in Multiplier Implementation," *IEEE Transactions on Computers*, pp. 1006–1015, Aug 1990.

[20] A. Gupta, "A 50ns 16×16 bit 2's complement parallel multiplier," in *ISELDECS*, 1987.

[21] E. L. Braun, *Digital Computer Design*, ch. 5. New York: Academic Press, 1963.

[22] G. W. McIver, R. W. Miller, and T. G. O. Shaughnessy, "A monolithic 16 × 16 digital multiplier," *IEEE Intenational Solid-State Circuits Conference Digest of Technical Papers*, pp. 231–233, 1974.

[23] D. P. Aggarwal, "High speed arithmetic arrays," *IEEE Transactions on Computers*, vol. C-28, pp. 215–224, March 1979.

[24] M. Nagamatsu, S. Tanaka, J. Mori, K. Hirano, T. Noguchi, and K. Hatanaka, "A 15ns 32 × 32 CMOS Multiplier with an Improved Parallel Structure," *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 494–497, April 1990.

[25] T. G. Noll, D. Schmitt-Landsiedel, H. Klar, and G. Enders, "A Pipelined 330-mhz Multiplier," *IEEE Journal of Solid-State Circuits*, vol. 21, pp. 411–416, June 1986.

[26] J. Y. lee, H. L. Garvin, and C. W. Slayman, "A High-Speed High-Density Silicon 8×8-bit Parallel Multiplier," *IEEE Journal of Solid-State Circuits*, vol. 22, pp. 35–39, Feb 1987.

[27] Y. Oowaki, K. Numata, K. Tsuchiya, K. Tsuda, and A. Hojo, "A Sub-10 ns 16×16 Multiplier Using 0.6μm CMOS Technology," *IEEE Journal of Solid-State Circuits*, vol. 22, pp. 762–766, Oct 1987.

[28] R. Sharma, A. D. Lopez, J. A. Michejda, S. J. Hillenius, J. M. Andrews, and A. J. Studwell, "A 16 × 16-bit Multiplier in Single-Level-metal CMOS Technology," *IEEE J. Solid-State Circuits*, vol. 24, pp. 922–926, Aug 1989.

[29] K. K. Primlani and J. L. Meador, "Nonredundant Radix-4 Serial Multiplier," *IEEE Journal of Solid-State Circuits*, vol. 24, pp. 1729–1735, Dec 1989.

[30] J. Mori, M. Nagamatsu, M. Hirano, S. Tanaka, M. Noda, Y. Toyoshima, K. Hashimoto, H. Hayashida, and K. Maeguchi, "A 10-ns 54 × 54-b Parallel Structured Full Array Multiplier with 0.5μm CMOS Technology," *IEEE Journal of Solid-State Circuits*, vol. 26, April 1991.

[31] K. K. Primlani and J. L. Meador, "A Pipelined 330-mhz Multiplier," *IEEE Journal of Solid-State Circuits*, vol. 24, Dec 1989.

[32] R. P. Brent and H. T. Kung, "The area-time complexity of binary multiplication," *Journal ACM*, vol. 28, pp. 521–534, July 1981.

[33] N. Takagi, H. Yasura, and S. Yajima, "High-Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree," *IEEE Transactions on Computers*, vol. c-34, pp. 789–796, Sept 1985.

[34] S. Vassiliadis, E. M. Schwarz, and D. J. Hanrahan, "A General Proof for Overlapped Multiple-Bit Scanning Multiplication," *IEEE Transactions on Computers*, vol. 38, pp. 172–183, Feb 1988.

Bibliography

Aggarwal, D. P., "High speed arithmetic arrays," *IEEE Transactions on Computers*, vol. C-28, pp. 215–224, March 1979.

Anderson, F. S., Earle, J. G., Goldschmidt, R. E., and Powers, D., "The IBM system 360/91 floating point execution unit," *IBM Jounal*, vol. 11, pp. 34–53, Jan 1967.

Bedrij, O. J., "Carry select adders," *IRE Transactions Electronics Computers*, vol. EC-11, pp. 65–73, June 1962.

Booth, A. D., "A signed binary multiplication technique," *Quaterly Journal of Mechanics and Applied Mathematics*, vol. 4, pp. 236–240, 1951.

Braun, E. L., *Digital Computer Design*, ch. 5. New York: Academic Press, 1963.

Brent, R. P. and Kung, H. T., "The area-time complexity of binary multiplication," *Journal ACM*, vol. 28, pp. 521–534, July 1981.

Brumm, P. and D.Brumm, *80386 A Programming and Design Handbook*, pp. 434–443. PA: TAB Books Inc., 1989.

Ghest, R. C., "A two's complement digital multiplier," Technical report, Advanced Micro Devices, Sunnyvale, CA, Nov 1971.

Gosling, J. B., "Design of a large high-speed floating-point-arithmetic unit," *IEE Proceedings*, 1971.

Gupta, A., "A 50ns 16×16 bit 2's complement parallel multiplier," in *ISELDECS*, 1987.

Habibi, A. and Wintz, P. A., "Fast multipliers," *IEEE Transactions on Computers*, pp. 153–157, Feb 1970.

Hutchison, D. and Silvester, P., *Computer Logic Principles and Technology*, pp. 173–190. England: Ellis Harwood Limited, 1987.

Hwang, K., *Computer Arithmetic Principles, Architecture and Design*, ch. 6. New York: Wiley, 1979.

lee, J. Y., Garvin, H. L., and Slayman, C. W., "A High-Speed High-Density Silicon 8×8-bit Parallel Multiplier," *IEEE Journal of Solid-State Circuits*, vol. 22, pp. 35–39, Feb 1987.

MacSorley, O. L., "High speed arithmetic in binary computers," in *Proceedings IRE*, Jan 1961.

McIver, G. W., Miller, R. W., and Shaughnessy, T. G. O., "A monolithic 16 × 16 digital multiplier," *IEEE Intenational Solid-State Circuits Conference Digest of Technical Papers*, pp. 231–233, 1974.

Mori, J., Nagamatsu, M., Hirano, M., Tanaka, S., Noda, M., Toyoshima, Y., Hashimoto, K., Hayashida, H., and Maeguchi, K., "A 10-ns 54 × 54-b Parallel Structured Full Array Multiplier with 0.5μm CMOS Technology," *IEEE Journal of Solid-State Circuits*, vol. 26, April 1991.

Nagamatsu, M., Tanaka, S., Mori, J., Hirano, K., Noguchi, T., and Hatanaka, K., "A 15ns 32 × 32 CMOS Multiplier with an Improved Parallel Structure," *IEEE Journal of Solid-State Circuits*, vol. 25, pp. 494–497, April 1990.

Noll, T. G., Schmitt-Landsiedel, D., Klar, H., and Enders, G., "A Pipelined 330-mhz Multiplier," *IEEE Journal of Solid-State Circuits*, vol. 21, pp. 411–416, June 1986.

Oowaki, Y., Numata, K., Tsuchiya, K., Tsuda, K., and Hojo, A., "A Sub-10 ns 16×16 Multiplier Using 0.6μm CMOS Technology," *IEEE Journal of Solid-State Circuits*, vol. 22, pp. 762–766, Oct 1987.

Primlani, K. K. and Meador, J. L., "Nonredundant Radix-4 Serial Multiplier," *IEEE Journal of Solid-State Circuits*, vol. 24, pp. 1729–1735, Dec 1989.

Primlani, K. K. and Meador, J. L., "A Pipelined 330-mhz Multiplier," *IEEE Journal of Solid-State Circuits*, vol. 24, Dec 1989.

Pucknell, D. A. and Eshraghian, K., *Principles of CMOS VLSI Design*, pp. 10–25. NJ: Prentice-Hall, 1988.

Rubinfield, L. P., "A proof of the modified Booth's algorithm for multiplication," *IEEE Transactions on Computers*, Oct 1975.

Sam, H. and Gupta, A., "A Generalized Multibit Recoding of two's Complement Binary Numbers and its Proof with Application in Multiplier Implementation," *IEEE Transactions on Computers*, pp. 1006–1015, Aug 1990.

Santoro, M. R. and Horowitz, M. A., "A Pipelined 64 × 64-bit Iterative Multiplier," *IEEE Journal of Solid-State Circuits*, vol. 24, pp. 487–493, April 1989.

Sharma, R., Lopez, A. D., Michejda, J. A., Hillenius, S. J., Andrews, J. M., and Studwell, A. J., "A 16 × 16-bit Multiplier in Single-Level-metal CMOS Technology," *IEEE J. Solid-State Circuits*, vol. 24, pp. 922–926, Aug 1989.

Shaw, R. F., "Arithmetic operations in a binary computer," *Rev. Science Instrumentation*, 1950.

Song, P. J. and Michelli, G. D., "Circuits and Architecture Trade-Offs for High-Speed Multiplication," *IEEE Journal of Solid-State Circuits*, vol. 26, pp. 1184–1192, Sept 1991.

Swartzlander, E. E., *Computer Arithmetic*, pp. 118–125. Vol. 1, Los Alamitos, CA: IEEE Computer Society Press, 1987.

Takagi, N., Yasura, H., and Yajima, S., "High-Speed VLSI Multiplication Algorithm with a Redundant Binary Addition Tree," *IEEE Transactions on Computers*, vol. c-34, pp. 789–796, Sept 1985.

Vassiliadis, S., Schwarz, E. M., and Hanrahan, D. J., "A General Proof for Overlapped Multiple-Bit Scanning Multiplication," *IEEE Transactions on Computers*, vol. 38, pp. 172–183, Feb 1988.

Wallace, C. S., "A suggestion for a fast multiplier," *IEEE Transactions on Computers*, vol. EC-13, Feb 1964.

Waser, S. and Flynn, M. J., *Introduction to Arithmeic for Digital System Designers*, pp. 156–157. New York: CBS Publishing, 1982.